

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри

О.В. Коваль

(підпис)

(ініціали, прізвище)

“ ” _____ 2019р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напрямку підготовки 6.050101 “Комп’ютерні науки”

на тему: «Задача порядку розроблення техніко-економічного обґрунтування систем освітлення для промислових споживачів»

Виконав: студент IV курсу, групи ТМ-51

Дубецький Андрій Станіславович

(прізвище, ім’я, по батькові)

(підпис)

Керівник кандидат технічних наук, доцент, Мамалига В. М.

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

(підпис)

Консультант _____

(назва розділу)

(вчені ступінь та звання, прізвище, ініціали)

(підпис)

Рецензент _____

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

(підпис)

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів без
відповідних посилань.

Студент _____

(підпис)

Київ – 2019

**Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”**

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрямок підготовки 6.050101 “Комп’ютерні науки”

ЗАТВЕРДЖУЮ
Завідувач кафедри
О.В. Коваль
(підпис)
” ” _____ 2018р.

ЗАВДАННЯ
на дипломну роботу студенту
Дубецькому Андрію Станіславовичу
(прізвище, ім’я, по батькові)

1. Тема роботи: «Задача порядку розроблення техніко-економічного обґрунтування систем освітлення для промислових споживачів»
керівник роботи Мамалига Володимир Михайлович, кандидат технічних наук, доцент
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ” ” 201__р. №

2. Строк подання студентом роботи _____
3. Вихідні дані до роботи мови програмування JavaScript, C#, середовище Visual Studio 2019, фреймворки .NET Core, ASP.NET Core, Entity Framework Core, React
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати існуючі рішення задачі техніко-економічного обґрунтування систем освітлення, розробити веб-додаток для підрахунку витрат на освітлення з можливістю вибору лампочок за критеріями, підрахунку необхідної кількості лампочок та виводу отриманої інформації у графічному виді
5. Перелік ілюстративного матеріалу актуальність, функції системи, використані формули, використані програмні засоби, архітектура бази даних, графічне представлення інтерфейсу
- _____
- _____
- _____

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|--------|-------------------------------------------|----------------|------------------|
| | | завдання видав | завдання прийняв |
| | | | |

7. Дата видачі завдання ” ____ ” _____ 2018 р.

КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів виконання дипломної роботи | Термін виконання етапів роботи | Примітки |
|-------|-----------------------------------------------------|--------------------------------|----------|
| 1. | Затвердження теми роботи | | |
| 2. | Вивчення та аналіз задачі | 05.02 – 11.02.19 | |
| 3. | Розробка архітектури та загальної структури системи | 12.02 – 18.02.19 | |
| 4. | Розробка структур окремих підсистем | 19.02 – 25.02.19 | |
| 5. | Програмна реалізація системи | 26.02 – 6.03.19 | |
| 6. | Оформлення пояснювальної записки | 12.03 – 30.05.19 | |
| 7. | Захист програмного продукту | 14.05.19 | |
| 8. | Передзахист | 28.05.19 | |
| 9. | Захист | | |

Студент _____
(підпис)

Керівник роботи _____
(підпис)

Дубецький А. С.
(прізвище та ініціали,)

Мамалига В. М.
(прізвище та ініціали,)

АНОТАЦІЯ

Дипломну роботу виконано на 56 аркушах, вона містить 3 додатки та перелік посилань на використані джерела з 19 найменувань. У роботі наведено 10 рисунків та 2 таблиці.

Метою даної дипломної роботи є розроблення техніко-економічного обґрунтування систем освітлення для промислових споживачів. Створений програмний продукт повинен бути реалізований веб-технологіями для легкого доступу та крос-браузерності. Крім того, він повинен мати зручний та інтерактивний інтерфейс користувача.

Ключові слова: вартість лампочок, техніко-економічне обґрунтування, економія електроенергії, системи освітлення, промислове освітлення.

ABSTRACT

The thesis is completed on 56 sheets, it contains 3 applications and a list of references to used sources of 19 titles. The paper presents 10 figures and 2 tables.

The purpose of this thesis is to develop a feasibility study for lighting systems for industrial consumers. The software product you create should be implemented with web technologies for easy access and cross-browser. In addition, it must have a user-friendly and interface.

Key words: cost of light bulbs, feasibility study, electricity saving, lighting systems, industrial lighting.

ЗМІСТ

| | |
|--------------------------------------------------------------------------------------|----|
| Вступ..... | 7 |
| 1 Постановка задачі..... | 8 |
| 2 Загальний аналіз проблем техніко-економічного обґрунтування систем освітлення | 9 |
| 2.1 Техніко-економічне обґрунтування | 9 |
| 2.2 Внутрішня норма прибутку..... | 11 |
| 2.3 Підрахунок вартості використання освітлення..... | 15 |
| 2.3.1 Підрахунок вартості використання освітлення в існуючих рішеннях | 15 |
| 2.3.2 Підрахунок вартості використання освітлення за допомогою IRR | 15 |
| 2.4 Знаходження необхідної кількості ламп..... | 17 |
| 2.5 Висновки до розділу | 19 |
| 3 Засоби розробки..... | 20 |
| 3.1 База даних MsSQL..... | 20 |
| 3.2 Мова програмування C# | 27 |
| 3.3 Технологія .NET Core | 28 |
| 3.4 Технологія ASP.NET Core..... | 29 |
| 3.5 Технологія Entity Framework Core..... | 30 |
| 3.6 Мова програмування JavaScript..... | 31 |
| 3.7 Фреймворк React | 36 |
| 3.8 Формальна мова CSS | 38 |
| 3.9 Висновки до розділу | 39 |
| 4 Опис програмної реалізації | 40 |
| 4.1 Опис архітектури проекту | 40 |
| 4.1.1 Архітектурний шаблон проектування MVC | 40 |
| 4.1.2 Архітектура веб-системи..... | 41 |
| 4.1.3 Концепт інверсії управління | 43 |
| 4.2 Опис структури проекту | 45 |

| | |
|--------------------------------------------------|----|
| 4.3 Висновки до розділу | 47 |
| 5 Робота користувача з програмною системою | 48 |
| 5.1 Інсталяція та системні вимоги | 48 |
| 5.2 Робота користувача з системою..... | 49 |
| 5.3 Висновки до розділу | 53 |
| Висновки | 54 |
| Перелік посилань..... | 55 |
| Додаток 1 | 57 |
| Додаток 2 | 59 |
| Додаток 3 | 69 |

ВСТУП

Проблема техніко-економічного обґрунтування систем освітлення для промислових споживачів є досить актуальною у зв'язку з необхідністю вибору лампочок, використання яких буде відповідати нормам освітленості приміщення та мати найбільш вигідну ціну.

В мережі інтернет можна знайти велику кількість ресурсів, які дозволяють розрахувати вартість використання лампочки, але навіть найбільш вдалі з них мають велику кількість недоліків:

- Неврахування важливих параметрів при підрахунку, наприклад відсотку вартості грошей, або час роботи на добу;
- Порівняння тільки двох ламп;
- Відсутність графічного відображення отриманих результатів;
- Неможливість перегляду отриманих результатів за певний проміжок
- Неможливість отримання результату відносно проміжку значень

Для реалізації техніко-економічного обґрунтування систем освітлення для промислових споживачів було створено веб-додаток, серверна частина якого написана на мові програмування C# з використанням технологій .NET Core, ASP.NET Core та Entity Framework Core, клієнтська частина була написана на мові програмування JavaScript з використанням фреймворку React, база даних - MsSQL. Використані технології є стандартом веб-розробки, що дозволяє запускати веб-додаток безпосередньо в браузері користувача й не потребує обов'язкового встановлення додаткового програмного забезпечення.

Цільовою галуззю застосування створеного веб-додатку є підрахунок вартості використання систем освітлення, а саме знаходження необхідної кількості лампочок та підрахунок вартості їхнього використання.

1 ПОСТАНОВКА ЗАДАЧІ

Метою даної дипломної роботи є реалізація програмного засобу, який проводить техніко-економічне обґрунтування систем освітлення для промислових споживачів.

Необхідними можливостями, які має забезпечувати застосунок, є:

- а) Реєстрація та авторизація користувача
- б) Вибір лампочки для розрахунку з наявного списку
- в) Підрахунок необхідного світлового потоку для заданого приміщення
- г) Підрахунок необхідної кількості ламп залежно від отриманого світлового потоку
- д) Підрахунок вартості використання лампи відносно часу роботи на добу, відсотку вартості грошей, тарифу та часу за який проводяться розрахунки
- е) Графічне зображення отриманих результатів
- ж) Редагування, видалення та додавання лампочок згідно з правами користувача.

2 ЗАГАЛЬНИЙ АНАЛІЗ ПРОБЛЕМ ТЕХНІКО-ЕКОНОМІЧНОГО ОБҐРУНТУВАННЯ СИСТЕМ ОСВІТЛЕННЯ

2.1 Техніко-економічне обґрунтування

Техніко-економічне обґрунтування – це оцінка практичності представленого системи або проекту.

Техніко-економічне обґрунтування має на меті раціонально та об'єктивно розкрити сильні та слабкі місця представленого проекту або системи, загрози і переваги, які існують у заданому середовищі, ресурси, які необхідні для успішного функціонування системи або проекту. Головним критерієм для оцінювання доцільності проекту або системи є співвідношення необхідних витрат та вартості, яку необхідно досягти.

Найбільш детальне техніко-економічне обґрунтування має містити опис наданого товару або послуги, історичний перелік заданої системи або проекту, деталі операцій та управління, маркетингові дослідження, фінансові дані, юридичні вимоги, податкові зобов'язання та бухгалтерську звітність.

Техніко-економічне обґрунтування оцінює потенціал проекту або системи та дозволяє оцінити шанси на успіх представленого бізнесу або підприємства. Об'єктивність дослідження є одним з самих важливих факторів довіри до проведеного дослідження для потенційних інвесторів та кредитних установ, тому техніко-економічне обґрунтування повинно проводитися з об'єктивним, неупередженим підходом для подання інформації, на основі якої можуть базуватися подальші рішення.

Техніко-економічне обґрунтування проекту або системи – це всебічний звіт, який детально розглядає п'ять основних структур аналізу представленого проекту. Також він враховує його чотири Р, точки вразливості та ризики, а також такі

обмеження, як календарні, фінансові та якісні. Головна ціль техніко-економічного обґрунтування проекту або системи полягає в визначені, чи повинен проект або система залишатись незмінним, бути переробленим, або завершити своє функціонування.

П'ять структур аналізу: структура визначення, структура контекстних ризиків, структура потенціалу, параметрична структура та структура стратегії домінуючих і непередбачених обставин.

Чотири Р розшифровуються як план (Plan), процеси (Processes), люди (People) та сила (Power). Зовнішні ризики, наприклад несприятливі погодні умови, поділені на вісім наступних категорій: фінансові та організаційні (планові), екологічні та технологічні (процесні), маркетинг і соціум (людські) та правові і політичні (сили). Точки вразливості – це внутрішні ризики проекту, які є контрольованими, або можуть бути усунені.

Обмеження (календарні, фінансові, якісні) можуть бути об'єктивно визначені та обчислені протягом всього життєвого циклу проекту або системи. В залежності від проекту або системи, кожна частина дослідження може бути достатньою для проведення техніко-економічного обґрунтування. Наприклад, невеликі проекти можуть не потребувати вичерпної екологічної оцінки.

Дослідження ринку – це один з найважливіших розділів техніко-економічного обґрунтування, оскільки він вивчає конкурентоспроможність наданих продуктів або послуг і переконує користувача у існуванні потенційного ринку для наданих продуктів або послуг. Якщо неможливо встановити ринок для наданих продуктів або послуг, то їх існування не має сенсу.

Як правило, ринкові дослідження оцінюють потенційні продажі продукту або послуг, коефіцієнт можливого поглинання та швидкості захоплення ринку, а також терміни реалізації проекту або системи.

Техніко-економічне обґрунтування представляє звіт з деталізацією критеріїв оцінки, результатами дослідження та рекомендації по покращенню роботи проекту або системи.

2.2 Внутрішня норма прибутку

Внутрішня норма прибутку (IRR) – це показник прибутковості інвестиції. Цей розрахунок виключає зовнішні фактори, такі як інфляція, вартість капіталу та різні фінансові ризики.

Внутрішня норма прибутку інвестиції є середньою геометричною точкою очікуваних прибутків вказаних інвестицій, що передбачає можливість зробити реінвестицію. Простіше кажучи, внутрішня норма прибутку – це ставка дисконту, при якій чиста поточна вартість або чиста приведена вартість дорівнює нулю.

Внутрішня норма прибутку може використовуватися як показник прибутковості проекту: чим вища внутрішня норма прибутку, тим вища загальна прибутковість, тому вона використовується як один з ключових критеріїв при прийнятті або відхиленні розглянутого інвестиційного проекту. Для цього внутрішня норма прибутку порівнюється з мінімальною ставкою або граничною ставкою, альтернативною вартості інвестиції, тобто якщо інвестиція не має ризику, альтернативні витрати, використані для порівняння внутрішньої норми прибутку, будуть безризиковою нормою прибутку. Якщо норма прибутковості проекту, виражена внутрішньою нормою прибутку, перевищує ставку відсікання, інвестиція приймається, інакше – відхиляється.

Великі корпорації використовують внутрішню норму прибутку у бюджеті для порівняння прибутковості масштабних проектів з точки зору норми прибутку. Наприклад, корпорація буде порівнювати інвестиції в нове підприємство у порівнянні

з розширенням існуючого підприємства на основі внутрішньої норми прибутку кожного з випадків. Чим вище внутрішня норма прибутку проекту, тим більш вигіднішим є виконання проекту, тобто щоб максимізувати віддачу, проект з найвищим рівнем внутрішньої норми прибутку буде вважатися самим пріоритетним тобто здійснюватися в першу чергу.

Внутрішня норма прибутку є показником прибутковості, ефективності, якості інвестицій, на відміну від чистої теперішньої вартості, яка є показником чистої вартості або величини, доданої шляхом здійснення нових інвестицій.

При застосуванні методу внутрішньої ставки доходу для максимізації вартості підприємства, будь-яка інвестиція буде прийнята, якщо її прибутковість, підрахована за допомогою внутрішньої норми прибутку, перевищує мінімальну норму прибутку. Відповідна мінімальна ставка для максимізації доданої вартості підприємства – це вартість капіталу, тобто внутрішня норма прибутковості нового масштабного проекту повинна бути вищою, ніж вартість капіталу компанії. Це пояснюється тим, що тільки інвестиція з внутрішньою нормою прибутку, яка перевищує вартість капіталу, має позитивну чисту приведену вартість.

Проте, вибір інвестицій може підлягати бюджетним обмеженням, або можуть існувати взаємовиключні конкуруючі проекти, або здатність керувати більшістю проектів можуть бути практично обмежені. У наведеному вище прикладі корпорації, яка обирала між інвестицією в нове підприємство та розширенням існуючого підприємства, може статися так що компанія не буде приймати участь в обох проектах.

Корпорації використовують внутрішню норму прибутку для оцінки необхідності випуску акцій та програм по викупу акцій. Викуп акцій відбувається, якщо повернення капіталу акціонерам має вищу внутрішню норму прибутку, ніж масштабні інвестиційні проекти або проекти купівлі за поточними ринковими цінами. Фінансування нових проектів шляхом залучення нових боргових зобов'язань може

також включати вимірювання вартості нового боргу з точки зору дохідності до погашення.

Враховуючи набір пар (час та грошовий потік), який представляє проект, чиста поточна вартість – це функція відсоткової ставки. Внутрішня норма прибутку – це норма, при якій ця функція дорівнює нулю, тобто внутрішня норма повернення є рішенням рівняння $NPV = 0$.

$$NPV = \sum_{j=0}^n \frac{C_j}{(1+IRR)^j}, \quad (2.1)$$

Де NPV – чиста поточна вартість;

n – час;

C_j – оплата через j років;

IRR – внутрішня норма прибутку.

У цій формулі C_0 є початковою інвестицією на самому початку проекту. Період відображення j зазвичай задається в роках, але обчислення може бути спрощеним, якщо IRR обчислюється з використанням періоду, в якому визначена більшість проблеми. Наприклад можна використовувати 1000 годин, якщо більшість грошових потоків будуть відбуватися через інтервал в 1000 годин.

У випадку, коли грошові потоки є випадковими величинами, очікувані значення вводяться у формулу (2.1).

У багатьох випадках значення внутрішня норма прибутку, що задовольняє рівнянню (2.1), не може бути знайдено аналітично, у цих випадках варто використовувати чисельні методи або графічні методи.

Як інструмент, що застосовується для прийняття інвестиційного рішення, показує, чи додає проект в своїй вартості або ні, порівнюючи внутрішню норму прибутку з необхідною нормою прибутку одного проекту, не розглядаючи будь-які інші проекти. Якщо відповідна внутрішня норма прибутку перевищує необхідну

норму прибутку, то, використовуючи необхідну норму прибутку для дисконтування грошових потоків до їх поточної вартості, NPV цього проекту буде позитивним, і навпаки. Однак, використання IRR для сортування проектів у порядку переваги не призводить до того ж порядку, що й NPV.

Однією з можливих інвестиційних цілей є максимізація загальної чистої поточної вартості проектів. Коли метою є максимізація загальної вартості, розрахована внутрішня норма прибутку не повинна використовуватися для вибору між взаємовиключними проектами. У тих випадках, коли один проект має більш високі початкові інвестиції, ніж другий взаємовиключний проект, перший проект може мати нижчу IRR (очікувану прибутковість), але більш високу NPV (збільшення багатства акціонерів) і, отже к розробці має бути прийнятий другий проект.

Коли метою є максимізація загальної вартості, внутрішню норму прибутку не слід використовувати для порівняння проектів різної тривалості. Наприклад, чиста поточна вартість, додана проектом з більшою тривалістю, але меншою внутрішньою нормою прибутку, може бути більшою, ніж у проекту подібного розміру, з точки зору загальних чистих грошових потоків, але з меншою тривалістю та вищою внутрішньою нормою прибутку.

Незважаючи на значну академічну прихильність NPV, різні дослідження показують, що більшість керівників віддають перевагу IRR. Очевидно, менеджерам легше порівняти інвестиції різних розмірів у відсотках доходності, ніж у грошах NPV. Однак NPV залишається більш точним відображенням цінності для бізнесу. IRR, як міра ефективності інвестицій, може дати краще уявлення про обмеженість капіталу. Однак при порівнянні взаємовиключних проектів внутрішня норма прибутку є найбільш робочим заходом для максимізації вартості.

2.3 Підрахунок вартості використання освітлення

2.3.1 Підрахунок вартості використання освітлення в існуючих рішеннях

В мережі інтернет можна знайти велику кількість ресурсів які дозволяють обчислити вартість використання освітлення, більшість з них використовує формулу (2.2).

$$C = \sum_1^n (p + P \times t \times 1000) \quad (2.2)$$

де C – вартість використання системи освітлення, грн;

n – кількість розрахункових годин, $1000 \times \text{год}$;

p – вартість лампочки ($p=0$, якщо нова лампа не потрібна), грн;

P – потужність лампи, кВт;

t – тариф, грн/кВт.год.

Оскільки формула (2.2) не враховує час роботи на добу, відсоток вартості грошей та кількість лампочок, вона не дає необхідні та достатньо точні результати.

2.3.2 Підрахунок вартості використання освітлення за допомогою IRR

Оскільки система освітлення – проект, який має свою вартість, то до нього можна застосувати внутрішню норму прибутку, тобто формула (2.1) прийме наступний вид:

$$C = \sum_{j=0}^n \frac{p+P \times t \times 1000}{(1+IRR)^j}, \quad (2.3)$$

де C – вартість використання лампочки, грн;

N – кількість лампочок, шт.;

n – кількість розрахункових годин, $1000 \times \text{год}$;

p – вартість лампочки ($p=0$, якщо нова лампа не потрібна), грн;

P – потужність лампи, кВт;

t – тариф, грн/кВт.год.

$$IRR = (1 + k)^{\frac{1000 \times 24}{8760 \times n}} - 1, \quad (2.4)$$

де k – відсоток вартості грошей;

n – час роботи напротязі доби.

Оскільки нульова ітерація – це етап інвестиції (купівля лампочки), а остання ітерація не потребує покупки нової лампочки, лампочок може бути декілька та після підстановки формули (2.4), формула (2.3) буде мати наступний вид:

$$C = N \times \left(p + \sum_{j=1}^{n-1} \frac{p+P \times T \times 1000}{(1+k)^{\frac{1000 \times 24 \times j}{8760 \times t}}} + \frac{P \times T \times 1000}{(1+k)^{\frac{1000 \times 24 \times n}{8760 \times t}}} \right), \quad (2.5)$$

де C – вартість використання системи освітлення, грн;

N – кількість лампочок, шт.;

n – кількість розрахункових годин, $1000 \times \text{год}$;

p – вартість лампочки ($p=0$, якщо нова лампа не потрібна), грн;

P – потужність лампи, кВт;

T – тариф, грн/кВт.год;

k – відсоток вартості грошей;

t – час роботи напротязі доби.

Отримана формула (2.5), у відмінності від формули (2.2), враховує всі важливі критерії, отже вона є більш приємною для розрахунку вартості використання систем освітлення.

2.4 Знаходження необхідної кількості ламп

Необхідний світловий потік знаходиться за формулою (2.6)

$$F_{\phi} = \frac{S \times k_3 \times z \times E_{min}}{\eta}, \quad (2.6)$$

де S – площа приміщення, m^2 ;

k_3 – коефіцієнт запасу;

z – коефіцієнт нерівномірності;

E_{min} – мінімальна нормована освітленість;

η – коефіцієнт використання світлового потоку;

Отже необхідну кількість ламп можна знайти за допомогою формули (2.7)

$$N = \frac{S \times k_3 \times z \times E_{min}}{F_{\lambda} \times \eta}, \quad (2.7)$$

де F_{λ} – світловий потік лампи, лм.

Коефіцієнт запасу знаходиться за допомогою таблиці (2.1).

Таблиця 2.1

| Приміщення | Коефіцієнт запасу k | | |
|-----------------------------------------------|--------------------------|------------------------|--------------|
| | Газоразрядні лампочки | Лампочки розжарення | Світлодіодні |
| Запиленість більше 5 мг/м ³ | 2 | 1,7 | 1,5 |
| Дим, копоть 1-5 мг/м ³ | 1,8 | 1,5 | 1,3 |
| Менше 1 мг/м ³ | 1,5 | 1,3 | 1,1 |
| Запиленість значано менше 1 мг/м ³ | 1,4 | 1,5 | 1 |

Коефіцієнт використання світлового потоку можна визначити за допомогою таблиці (2.2)

Таблиця 2.2

| | | | | | |
|-------------|--------------------------------|------|------|------|------|
| $\rho_{сл}$ | 0,7 | 0,7 | 0,5 | 0,5 | 0 |
| $\rho_{ст}$ | 0,5 | 0,5 | 0,5 | 0,3 | 0 |
| $\rho_{п}$ | 0,3 | 0,1 | 0,1 | 0,1 | 0 |
| i | Коефіцієнт використання η | | | | |
| 0,5 | 0,23 | 0,20 | 0,20 | 0,17 | 0,10 |
| 0,6 | 0,28 | 0,26 | 0,24 | 0,20 | 0,14 |
| 0,7 | 0,32 | 0,30 | 0,28 | 0,24 | 0,17 |
| 0,8 | 0,35 | 0,33 | 0,30 | 0,26 | 0,19 |
| 0,9 | 0,38 | 0,35 | 0,33 | 0,29 | 0,21 |
| 1,0 | 0,41 | 0,38 | 0,35 | 0,31 | 0,23 |
| 1,1 | 0,43 | 0,40 | 0,37 | 0,33 | 0,25 |
| 1,25 | 0,45 | 0,41 | 0,38 | 0,35 | 0,27 |
| 1,5 | 0,49 | 0,45 | 0,42 | 0,38 | 0,30 |
| 1,75 | 0,52 | 0,47 | 0,44 | 0,41 | 0,32 |
| 2,0 | 0,54 | 0,49 | 0,45 | 0,42 | 0,33 |
| 2,25 | 0,56 | 0,51 | 0,47 | 0,44 | 0,35 |
| 2,5 | 0,58 | 0,52 | 0,48 | 0,46 | 0,36 |
| 3,0 | 0,60 | 0,54 | 0,50 | 0,48 | 0,38 |
| 3,5 | 0,62 | 0,55 | 0,51 | 0,49 | 0,39 |
| 4,0 | 0,64 | 0,56 | 0,52 | 0,50 | 0,40 |
| 5,0 | 0,67 | 0,59 | 0,54 | 0,53 | 0,43 |

де ρ – коефіцієнти відображення стелі, стін та полу відповідно

i – індекс приміщення, знаходиться за допомогою формули (2.8)

$$i = \frac{a \times b}{(h - h_{\text{л}} - h_{\text{п}}) \times (a + b)}, \quad (2.8)$$

де a – довжина приміщення;

b – ширина приміщення;

h – висота приміщення;

$h_{\text{л}}$ – висота підвісу лампочки;

$h_{\text{п}}$ – висота робочої поверхні.

Отже для знаходження необхідної кількості лампочок необхідно знати світловий потік, ширина, висота та довжина приміщення, висота робочої поверхні, висота підвісу світильника, коефіцієнт відображення поверхні, коефіцієнт запасу, коефіцієнт нерівномірності та мінімальна нормована освітленість.

2.5 Висновки до розділу

В цьому розділі була розглянута проблематика техніко-економічного обґрунтування систем або проектів, проблематика внутрішньої норми прибутку, представлена формула, яка використовуються в існуючих рішеннях, виведена більш досконала формула з врахуванням внутрішньої норми прибутку та представлена формула знаходження необхідної кількості лампочок.

3 ЗАСОБИ РОЗРОБКИ

3.1 База даних MsSQL

Microsoft SQL Server – це система управління реляційними базами даних, перша версія якої була випущена компанією Microsoft 24 квітня 1989 року.

Використовувана мова розробки (за командним рядком або через графічний інтерфейс Management Studio) – Transact-SQL (TSQL), що є реалізацією стандарту ANSI мови SQL, що використовується для маніпулювання та відновлення даних (DML), створення таблиць і визначення відносини між ними (DDL).

Найбільш відомими альтернативами MsSQL є Oracle, MariaDB, MySQL та PostgreSQL. SQL Server був доступний тільки для операційних систем розроблених компанією Microsoft, але з 2016 року він доступний для Linux та з 2017 року для Docker.

База даних MsSQL може бути налаштована на використання декількох екземплярів на одному фізичному сервері, причому перша установка зазвичай несе в собі назву сервера, а всі наступні – конкретні імена (з інвертованим скриптом між ім'ям сервера і назвою установки).

База даних MsSQL має наступні особливості:

- підтримка транзакцій;
- підтримка збережених процедур;
- графічне середовище адміністрування;
- режим клієнт-сервер, де інформація і дані розміщуються на сервері, а клієнти мережі мають доступ до інформації;
- можливість керувати інформацією з інших серверів даних.

Ця система включає в себе скорочену версію, яка називається MSDE з тим же ядро бази даних, але орієнтована на менші проекти.

Загальноприйнятою є розробка проектів з використанням Microsoft SQL Server і Microsoft Access через так званий проект ADP (Access Data Project). Це дозволяє роботу між базою даних (Microsoft SQL Server) та середовищем розробки (VBA Access), за рахунок реалізації дворівневих додатків за допомогою форм Windows.

При обробці SQL за допомогою командних рядків використовується SQLCMD, osql або PowerShell.

Для розробки більш складних додатків (три або більше шарів) Microsoft SQL Server включає інтерфейси доступу для декількох платформ розробки, включаючи .NET, але сервер є доступним тільки для операційних систем.

Тип NUMERIC було покращено, щоб використовувати його як ідентифікатор стовпця у версії 2008 R2.

T-SQL (Transact-SQL) є основним засобом взаємодії з сервером, що дозволяє виконувати ключові операції в Microsoft SQL Server, включаючи створення і модифікацію схем бази даних, вставки і модифікації даних в базі даних, а також адміністрування сервера. Це робиться за допомогою надсилання операторів в T-SQL і операторів, які обробляються сервером, і результати або помилки повертаються до клієнтського додатка.

Клієнт Native SQL – це бібліотека доступу до даних для клієнтів Microsoft SQL Server. Він підтримує функції Microsoft SQL Server, включаючи виконання табличного потоку даних, підтримку баз даних дзеркал Microsoft SQL Server, повну підтримку всіх типів даних, що підтримуються в Microsoft SQL Server, асинхронні набори операцій, сповіщення про запити, підтримку шифрування, а також отримання декількох наборів результатів в одній сесії бази даних. Рідний SQL-клієнт використовується як розширення плагінів Microsoft SQL Server для інших технологій доступу до даних, включаючи ADO або OLE DB. Рідний SQL-клієнт також може використовуватися безпосередньо, мінаючи шари доступу до даних.

Кожна ітерація розвитку Microsoft SQL Server має різні версії з різною вартістю, які залежать від фізичної конфігурації сервера.

Версія Enterprise включає в себе всі функції (відключені в інших виданнях), це тип версії з найбільшими привілеями на ринку, підтримує необмежену кількість процесорів, а також агрегацію пам'яті без переривання служби або сервера, у той час як стандартна версія обмежена 16 процесорами.

Версія Developer – це видання з тими ж характеристиками, що й Enterprise, але може бути використана тільки в середовищі розробки, а не у виробництві. Якщо база даних була розроблена для стандартної версії, то необхідно врахувати відсутні у цій версії можливості.

Версія Standard обмежена відповідно до конфігурації сервера та його функцій, призначених для нижчих серверів.

Версія Express є безкоштовною, що дозволяє створювати обмежені бази даних з основним функціоналом, для підтримки додатків, які потребують простого рішення для зберігання обмеженого обсягу даних, або користувачів, чії ресурси та потреби обмежені, ця версія може використовувати максимум 1 Гб пам'яті та зберігати не більше 10 ГБ, працює на серверах з максимальною кількістю чотирьох процесорів.

Версія SQL Azure – це версія Microsoft SQL Server в хмарі, яка дозволяє щомісяця платити за послугу без необхідності підтримувати фізичний сервер. Компанія платить тільки за доступ, а доступ надається через сервери в різних кутках світу. За допомогою SQL Azure не потрібно встановлювати, підтримувати або оновлювати фізичний сервер, хоча ця послуга залежить від потреб компанії, пов'язаних з проблемами безпеки через те що сервер знаходиться поза компанією та необхідність в наявності швидкісного та безперебійного підключення до мережі інтернет.

Microsoft SQL Server надає деякі інтерфейси, які змінювалися протягом багатьох років, серед яких найбільш відомими є графічні інтерфейси, які використовуються як стандартний інструмент розробника для розробки і адміністрування.

Графічний інтерфейс до 2005 року включав Enterprise Manager з деревним переглядом різних об'єктів і з можливістю їх обробки; і аналізатор запитів як текстовий інтерфейс для виконання команд TSQL.

У 2005-му році ці два засоби були уніфіковані в один - SQL Server Management Studio (SSMS), а з 2008 року включена можливість роботи з Visual Studio - стандартний інтерфейс розробки Microsoft. Іншим необов'язковим інтерфейсом є використання командного рядка з інструментами, такими як SQLCmd, ISQL, OSQL, що дозволяє виконувати сценарії та пакетну обробку. З 2008 року він може бути розроблений за допомогою SQLCmd (SQL Command) через SSMS без взаємозв'язку з текстовим інтерфейсом Windows. Іншим варіантом в області сценаріїв є використання мови сценаріїв Microsoft Powershell.

Крім стандартних інтерфейсів Microsoft SQL Server, можна виконувати команди TSQL за допомогою інструментів підключення, таких як ODBC і OLE-DB.

На відміну від систем баз даних, таких як Microsoft Access, які є "пасивними" і містять файл для підключення і виконання команд, які здійснюється в клієнті (комп'ютері користувача), в Microsoft SQL Server є ряд послуг, які виконуються системою в пам'яті сервера що дозволяє використовувати всі можливості сервера, що є більш потужним та запобігають перевантаженню мережі і мають планувальник завдань, тобто завдання будуть виконуються навіть якщо клієнт не є активним.

Основні послуги:

- SQL Server – ядро системи;
- SQL Agent – виконання завдань (запрограмовані скрипти) та надсилання попереджень у разі навантажень у системі;
- повнотекстовий фільтр Daemon Launcher – використання в спеціальних індексах "Повного текстового пошуку" шляхом розширеного пошуку тексту;
- SQL Browser – "слухач", призначений для відправлених команд і перенаправлення їх до місця призначення;
- SSIS Server – робота SSIS (інструмент ETL);

- SSAS Server – робота SSAS (інструмент OLAP);
- Сервер SSRS – операція SSRS (інструмент звітності);

У кожній установці SQL Server є 4 системні бази даних, а також можливість створювати нові бази даних користувачем, в яких дані зберігаються в таблицях.

Ці бази даних, створені користувачами, в основному включають в себе файл даних (з розширенням mdf) з таблицями і різними об'єктами на рівні бази даних, файл журналу (з розширенням ldf) з відкритими транзакціями і закритими транзакції, тема до обраної моделі відновлення (всі зміни в базі даних можуть бути накопичені в файлі журналу з моменту останнього резервного копіювання). Можливість створити набір файлів даних на додаток до основного (з розширенням ndf) для розгляду ефективності, розділення навантаження між жорсткими дисками тощо.

Системні бази даних:

- master – всі процедури, функції і таблиці системи, які використовуються всіма базами даних і які встановлюються автоматично, а також ті, які були створені адміністраторами системи. Крім того, всі визначення щодо безпеки на рівні сервера зберігаються в цій базі даних.

- msdb – зберігання завдань агента, кодів CLR, об'єднаних у систему, пакетів SSIS та інших.

- Форма баз даних. Кожна нова база даних створюється як копія цієї бази даних, якщо не було чітко визначено інше.

- tempdb – тимчасова база даних, яка створюється знову кожного разу, коли служба перезапускається. Він використовується для зберігання тимчасових таблиць, створених користувачами або системою.

З логічної точки зору дані зберігаються в базах даних в таблицях, через які реалізується теорія реляційних баз даних. Таблиця розділена на рядки та стовпці (іноді їх називають записами та полями). Таблиці можуть бути фіксованими або тимчасовими, а у другому випадку вони фізично існують у базі даних tempdb, і вони

автоматично видаляються у разі відключення сеансу або з'єднання з сервером, залежно від типу тимчасової таблиці.

З фізичної точки зору, система розділяє файли баз даних на 64 Кб, а кожен з них на вісім 8 Кб сторінок. Як правило, кожен екстент присвоюється таблиці або індексу, за вирахуванням малих таблиць і кожна сторінка завжди призначена для певної таблиці. Система відповідає за збільшення файлів, відповідно до налаштувань користувача.

Можна створювати індекси до таблиць. Індеси зберігаються поруч із таблицею (Non Clustered Index) або є самою таблицею (Clustered Index). Індеси допомагають у пошуку даних у таблицях (наприклад, у файлах у бібліотеках), в їх упорядкуванні та визначенні первинних ключів.

Між таблицями можна створити один-до-багато відносин.

Окрім таблиць користувачів, є таблиці, в яких зберігаються метадані: дані про саму систему, різні об'єкти, права, статистику про роботу системи (DMV), тощо.

Для кожного стовпця таблиці та кожної змінної або параметра визначається тип даних, що зберігаються в ньому, включаючи:

Числа: цілі і не цілі числа різних розмірів і на різних рівнях точності; і додатковий автоматичний приріст.

Тексти: Ланцюги різної довжини та різні можливості для підтримки різних мов.

Дати: Дати в різних рівнях точності, від повного дня до дробів менше однієї секунди, які підтримують початок 20-го століття або григоріанського календаря, і здатність розрізняти різні види використання графіків.

XML: текстові дані (рядки), які представляють стандартні набори даних (стандарт SGML).

Двійкові дані: дані, що зберігаються як двійкові дані (біти і байти), які дозволяють зберігати графічні файли, тощо.

Географія: Стандартне представлення географічної інформації, наприклад, держав, географічних зон, населених пунктів, розрахункової відстані.

Геометрія: Стандартне представлення точок, ліній, поверхонь у площині; і відносини між ними.

Ієрархія: Стандартне представлення ієрархічної інформації, наприклад, перелік матеріалів, відносини підпорядкування між працівниками, тощо.

Процедури є командними скриптами TSQL, які можуть виконуватися з різними параметрами.

Збережені процедури можуть значно полегшити адміністрування бази даних і відображення інформації про зазначену базу даних і її користувачів. Збережені процедури являють собою попередньо скомпільований збірник операторів SQL і додаткові команди керування потоком, що зберігаються під одним ім'ям і обробляються як одиниця. Збережені процедури зберігаються в базі даних; вони можуть виконуватися з програми і дозволяти змінні, оголошені користувачем, умовне виконання та інші ефективні функції програмування. Збережені процедури можуть містити потік програми, логіку і запити до бази даних. Вони можуть приймати параметри, повертати один або декілька наборів результатів та повертати значення.

Можна процедуру один раз, зберегти її в базі даних і викликати її з програми стільки разів, скільки необхідно. Фахівець з програмування баз даних може створювати збережені процедури, які потім можуть бути змінені незалежно від вихідного коду програми. Вони полегшують обслуговування.

У ситуаціях, коли потрібна велика кількість коду Transact-SQL або якщо операції виконуються кілька разів, збережені процедури можуть бути швидше, ніж частини коду Transact-SQL. Процедури аналізуються і оптимізуються під час їх створення, і можна використовувати версію процедури, яка знаходиться в пам'яті після її першого виконання. Оператори Transact-SQL, які надсилаються клієнтом повинні бути зібрані і оптимізовані, коли SQL Server виконує їх.

Операція, яка потребує сотень рядків коду Transact-SQL, може бути виконана однією командою, яка виконує код у процедурі, замість того, щоб посилати сотні рядків коду через мережу.

Окрім альтернативних рішень на рівні операційної системи (резервне копіювання файлів баз даних), в SQL Server є інтегрований інструмент, який дозволяє здійснювати повну або диференціальну резервну копію відповідно до попередньо визначеної моделі відновлення (Recovery Model). Є можливість створити резервну копію через скрипт (з даними або без них). З 2008 року файли резервних копій можна стискати що дає змогу зменшити фізичні розміри таблиць і індексів і більш ефективно використовувати обсяг жорстких дисків.

Агент – це служба, відповідальна за завдання планування, і відповідає за їх виконання самостійно. Зазвичай він виконує завдання з технічного обслуговування, складні завдання ETL, резервні копії тощо.

SQL Server має можливість надсилати електронні листи через код. Ця функціональність, як правило, використовується для надсилання сповіщень про проблеми в системі (наприклад, якщо використання процесора збільшилося до попередньо визначеного порогу або якщо процес ETL не вдався), а також після успішного завершення процесу.

3.2 Мова програмування C#

C# – це об'єктно-орієнтована мова програмування, розроблена Андресом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде та стандартизована корпорацією Майкрософт як частина платформи .NET, яка пізніше була затверджена стандартом ECMA (ECMA-334) та ISO (ISO / IEC 23270). C# є однією з мов програмування, призначених для спільної мовної інфраструктури.

Основний синтаксис C# походить від C / C ++ і використовує об'єктну модель платформи .NET, подібну до Java, хоча вона включає в себе вдосконалення, отримані з інших мов.

Ім'я C Sharp було натхнене знаком '#', який складається з чотирьох знаків '+', що якби каже про те що C# є наступним етапом розвитку мови C++, яка в свою чергу була новим етапом розвитку C, але насправді C# і C++ мають великі відмінності через те що C++, навідмінно від C#, є низькорівневою мовою.

Мова має строгу типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості. Перейнявши багато від своїх попередників, C#, спираючись на практику їхнього використання, не використовує деякі моделі, що зарекомендували себе як проблемні при розробці програмних систем, наприклад множинне спадкування класів.

3.3 Технологія .NET Core

.NET Core – це модульна реалізація, яка може використовуватися у широкому спектрі застосувань, починаючи з дата-центрів і закінчуючи розумними годинниками, доступна з відкритим вихідним кодом, і підтримувана корпорацією Microsoft на трьох основних системах: Windows, Linux і Mac OSX. Перша версія .NET Core 1.0 була випущена 27 червня 2016 року.

.NET Core, як і .NET Framework, підтримує C#, Visual Basic та F#. На основі .NET Core було створено ASP.NET Core.

.NET Core – це .NET Framework, чия реалізація була оптимізована з точки зору завдань декомпозиції, тобто .NET Core є крос-платформною та має можливість застосовувати хмарні технології, відбувся поділ між бібліотекою CoreFX і середовищем виконання CoreCLR.

CoreFX – це бібліотека, інтегрована в .NET Core.

CoreCLR – це середовище виконання, що включає в себе RyuJIT (JIT-компілятор), вбудований збирач сміття та інші важливі компоненти.

.NET Core – модульна платформа, тобто кожен її компонент оновлюється через менеджер пакетів NuGet, а значить можна оновлювати її модулі окремо, на відміну від .NET Framework, який має оновлюється цілком, що викликало труднощі при переході на більш нову версію, тобто кожна програма може працювати з різними модулями і не залежати від єдиного оновлення.

3.4 Технологія ASP.NET Core

ASP.NET Core – це нове покоління ASP.NET, раніше відоме як ASP.NET vNext, і було названо ASP.NET 5 на початку запуску, через деякий час було перейменовано в .NET Core. ASP.NET Core це нове покоління функціональності ядра ASP.NET, яке було побудованого з нуля.

ASP.NET Core працює на трьох основних операційних системах: Windows, Mac OSX і Linux, і є першою платформою Microsoft для веб-розробки з крос-платформними можливостями.

Microsoft відкрила код ASP.NET Core на початку розробки, тому що воно також є учасником проекту з відкритим кодом, керованим .NET Foundation.

Технологія ASP.NET Core має наступні особливості:

- Модульна структура розподіляється за допомогою NuGet;
- Cloud-optimized runtime (оптимізована для Інтернету);
- Єдина історія для створення веб UI і веб APIs
- Система створення конфігурації середовища на основі хмарових технологій;
- Легкий і модульний HTTP запит;
- Створення та запуск крос-платформних додатків ASP.NET Core у Windows, Mac та Linux;
- Відкритий код та орієнтація на спільноту;

3.5 Технологія Entity Framework Core

Entity Framework Core – це легка, розширювана, відкрита і крос-платформна версія технології для роботи з базами даних Entity Framework.

Entity Framework Core може служити об'єктно-реляційним відображенням, що дозволяє розробнику працювати з базою даних за допомогою об'єктів .NET, а також виключає необхідність написання більшої частини коду для доступу до даних, яка була необхідна в Entity Framework та підтримує багато механізмів баз даних.

Entity Framework Core має універсальний API для роботи з даними, тобто якщо змінити цільову СУБД, то основні зміни в проекті будуть стосуватися насамперед конфігурації і налаштування підключення. А код, який працює з даними, отримує, додає, редагує та видаляє дані, залишиться колишнім.

У Entity Framework Core доступ до даних здійснюється за допомогою моделі, яка складається з класів сутностей і об'єкта контексту, який представляє сесію з базою даних, через що є можливість запитувати і зберігати дані.

Користувач має можливість генерувати модель з існуючої бази даних, що відповідає існуючій базі даних, або використовувати Entity Framework Migrations для створення бази даних з написаної моделі, а потім доповнювати або змінювати її протягом розвитку моделі.

У Entity Framework Core екземпляри класів сутностей витягуються з бази даних за допомогою Language Integrated Query (LINQ), дані створюються, редагуються та видаляються в базі даних за допомогою примірників класів сутностей, сутності можуть бути пов'язані асоціативною зв'язком один-до-багатьох, один-до-одного і багато-до-багатьох, подібно до того, як в реальній базі даних відбувається зв'язок через зовнішні ключі.

3.6 Мова програмування JavaScript

JavaScript є інтерпретованою мовою програмування, стандарту ECMAScript. Вона визначений як об'єктно-орієнтована, прототипна, імперативна, слабо типізована та динамічна мова програмування.

В основному JavaScript використовується на стороні клієнта, реалізованої як частина веб-браузера, що дозволяє поліпшити користувацький інтерфейс та зробити веб-сторінку більш динамічною, проте JavaScript може бути використаним на стороні сервера за допомогою інтерпретатора node.js. Також JavaScript використовується в додатках, які не відносяться до браузера, наприклад, у документах PDF, десктопних додатках (в основному віджети).

З 2012 року всі сучасні браузери повністю підтримують ECMAScript 5.1. Старі браузери підтримують принаймні ECMAScript 3. Шосте видання було випущено в липні 2015 року.

JavaScript був розроблений з синтаксисом, подібним до C, хоча він приймає імена та угоди мови програмування Java. Проте Java і JavaScript мають різну семантику і цілі.

Усі сучасні браузери інтерпретують інтегрований код JavaScript у веб-сторінках. Для взаємодії JavaScript з веб-сторінкою передбачена мова для реалізації об'єктної моделі документа (DOM).

Раніше JavaScript використовувався у веб-сторінках HTML для виконання операцій і тільки в рамках клієнтського додатка, без доступу до функцій сервера. В даний час JavaScript широко використовується для передачі та отримання даних з сервера разом з іншими технологіями, такими як AJAX. JavaScript інтерпретується в користувацькому агенті одночасно з завантаженням розмітки HTML.

З моменту випуску в червні 1997 року стандарту ECMAScript 1 були версії 2, 3 і 5, які в даний час найбільш часто використовуються. У червні 2015 року була

випущена актуальна версія ECMAScript 6, яка на даний час підтримується в усіх нових браузерах.

JavaScript був розроблений Бренданом Айхом з Netscape під назвою Mocha, згодом був перейменований в LiveScript, після чого в JavaScript. Зміна імені збігалася з часом, коли Netscape додав підтримку Java-технологій у своєму веб-браузері Netscape Navigator у версії 2.002 у грудні 1995 року. Назва викликала плутанину, створюючи враження, що мова JavaScript є розширенням мови Java, тому багато хто характеризує це як маркетингову стратегію Netscape, щоб отримати престиж та інновації в області нових мов веб-програмування.

"JAVASCRIPT" є зареєстрованим товарним знаком корпорації Oracle і використовується за ліцензією на продукти, створені компанією Netscape Communications і поточними організаціями, такими як Mozilla Foundation.

Microsoft надала своїй версії JavaScript ім'я JScript, для того щоб уникнути проблем, пов'язаних з авторськими правами на бренд. JScript був прийнятий у версії 3.0 Internet Explorer, випущеної в серпні 1996 року, і включав сумісність з Effect 2000. Мови можуть здаватися настільки схожими, що терміни "JavaScript" і "JScript" часто використовуються як взаємозамінні, але специфікація JScript у багатьох відношеннях несумісна з ЕСМА.

Щоб уникнути цих несумісностей, Консорціум Всесвітньої Павутини розробив стандартну Об'єктну Модель Документів (DOM або Document Object Model), що включає Konqueror, версії 6 Internet Explorer і Netscape Navigator, Opera 7, Mozilla Application Suite і Mozilla Firefox з першої версії.

У 1997 році автори запропонували JavaScript бути прийнятим як стандарт Європейської асоціації виробників комп'ютерів ЕСМА, яка, незважаючи на свою назву, не є європейською, а міжнародною і була заснована в Женеві. У червні 1997 року він був прийнятий як стандарт ЕСМА під назвою ECMAScript.

Netscape представила серверну реалізацію скриптів з Netscape Enterprise Server, випущеною у грудні 1994 року (незабаром після випуску JavaScript для веб-

браузерів). Починаючи з середини 2000-х років, існує поширення реалізацій JavaScript на стороні сервера. Node.js є одним з помітних прикладів JavaScript на серверній стороні, що використовується у важливих проектах.

JavaScript став одним з найбільш популярних мов програмування. Але на початку багато розробників відмовлялися від використання мови. Внаслідок цього відбулося поширення набору фреймворків і бібліотек загального масштабу, поліпшення практики програмування за допомогою JavaScript і збільшення використання JavaScript за межами веб-браузерів, як це видно з поширення середовищ JavaScript на стороні сервера. У січні 2009 року проект CommonJS був відкритий з метою визначення бібліотеки для використання загальних завдань, головним чином для розробки поза веб-браузером.

У червні 2015 року був опублікований стандарт ECMAScript 6 з нерегулярною підтримкою серед браузерів, забезпечуючи JavaScript розширеним функціоналом, який є в інших мовах програмування, таких як модулі для організації коду, істинних класів для об'єктно-орієнтованого програмування, стрілочних функцій, ітераторів, генераторів або обіцянок для асинхронного програмування, проте більшість з них є звичайним синтаксичним цукром, який суттєво перетворює код в кращу сторону, зробивши його більш ємким та зрозумілим.

JavaScript сумісний з більшою частиною структури мови програмування C, але в C, область видимості змінних не виходить за рамки блоку, в якому вони були визначені, а в JavaScript область видимості змінних знаходиться в рамках функції, в якій вони були оголошені. Однак це змінилося з виходом ECMAScript 2015, оскільки вона додає підтримку для блочного визначення області видимості за допомогою оператора `let`. Як і C, JavaScript розрізняє функції та процедури. Синтаксична відмінність по відношенню до C – це автоматична вставка крапки з комою, тобто в JavaScript крапка з комою, може бути опущена, проте краще все таки її ставити.

Оскільки JavaScript є нетипізованою мовою, у змінній нема прив'язки до певного типу, наприклад, змінна `x` в даний момент може бути числом а пізніше –

рядком. JavaScript підтримує кілька способів перевірки типу змінної, в тому числі і типізацію. Один із способів дізнатися - це ключове слово `typeof`.

JavaScript майже повністю складається з об'єктів. Об'єкти в JavaScript є асоціативними масивами, поліпшеними з включенням прототипів. Імена властивостей об'єктів - це ключі типу `string`: `obj.x = 10` і `obj['x'] = 10` еквівалентні, що є синтаксичним цукром, позначенням з точкою. Властивості та їх значення можна створювати, змінювати або видаляти під час виконання. Більшість властивостей об'єкта (і тих, які включені ланцюгом прототипичного наслідування) можна перерахувати за допомогою циклу `for in`. У JavaScript є невелика кількість заздалегідь визначених об'єктів, таких як функція та дата.

Функції самі по собі є об'єктами. Як такі, вони мають властивості і методи, такі як `.call()` або `.bind()`. Вкладена функція є функцією, визначеною в межах іншої. Так виходить, коли викликається зовнішня функція. Крім того, кожна створена функція утворює замикання, що містить одну або більше залежних змінних з іншого зовнішнього середовища, включаючи константи, локальні змінні та аргументи зовнішньої функції виклику. Результат оцінки згаданого замикання є частиною внутрішнього стану кожної об'єктної функції, навіть після того, як зовнішня функція завершує свою роботу.

JavaScript використовує прототипи замість класів для використання успадкування, можна емулювати багато функцій, що представляються класами в традиційних об'єктно-орієнтованих мовах, через прототипи в JavaScript.

Функції також ведуть себе як конструктори. Префікс виклику функції з ключовим словом `new` створює новий екземпляр прототипу, успадковуючи властивості і методи від конструктора (включаючи властивості прототипу `Object`) ECMAScript 5 пропонує метод `Object.create`, що дозволяє явне створення екземпляра без необхідності успадковувати автоматично від прототипу `Object` (у старих середовищах може з'явитися прототип об'єкта, створеного як `null`.) Властивість прототипу конструктора визначає об'єкт, що використовується для внутрішнього

прототипу нових створених об'єктів. Нові методи можуть бути додані шляхом зміни прототипу об'єкта, що використовується як конструктор. Конструктори, визначені в JavaScript, такі як Array або Object, також мають прототипи, які можна змінювати. Хоча зміна прототипу об'єкта вважається поганою практикою, оскільки більшість об'єктів у Javascript успадковують методи і властивості від об'єкта-прототипу.

JavaScript залежить від середовища, в якому воно виконується (наприклад, у веб-браузері), щоб запропонувати об'єкти та методи, за допомогою яких скрипти можуть взаємодіяти з "зовнішнім світом".

До функції може бути передано невизначене число параметрів. Функція може отримати доступ до них через параметри або також через аргументи локальних об'єктів.

На відміну від багатьох об'єктно-орієнтованих мов, в JavaScript не існує різниці між визначенням функції та визначенням методу, проте різниця є під час виклику функції. Функцію можна об'явити методом. Коли функція є методом об'єкта, ключове слово `this`, яке є локальною змінною до функції, являє собою об'єкт, який викликав цю функцію.

JavaScript, як і більшість мов програмування, підтримує регулярні вирази, які забезпечують стислий і потужний синтаксис для маніпулювання строками, і є більш складним, ніж функції, вбудовані в строкові об'єкти.

Найбільш поширене використання JavaScript – написання функцій, які взаємодіють з об'єктною моделлю документа (DOM) на сторінці.

Оскільки код JavaScript може бути виконаний локально в браузері користувача, а не на віддаленому сервері, браузер може миттєво реагувати на дії користувача, роблячи веб-сторінку більш чуйною. З іншого боку, JavaScript-код може виявляти дії користувача, які HTML не бачить, наприклад, натискання на клавіші. Такі сервіси, як Gmail, користуються цією перевагою.

Веб-браузер є найпоширенішою середовищем для інтерпретатора JavaScript. Веб-браузери зазвичай створюють об'єкти, залежні від середовища виконання, для

представлення об'єктної моделі документа (DOM) у JavaScript. Веб-сервер – це інше звичайне сервісне середовище. Веб-сервер JavaScript зазвичай виставляє свої власні об'єкти для представлення об'єктів HTTP-запиту і відповіді, якими інтерпретатор JavaScript може маніпулювати для динамічного створення веб-сторінок.

Оскільки JavaScript – це єдина мова для написання клієнтської частини, в неї компілюються більшість фреймворків на інших мовах, хоча JavaScript не був розроблений для таких цілей.

3.7 Фреймворк React

React – це фреймворк написаний на мові програмування Javascript з відкритим вихідним кодом. React призначений для створення чуйних користувацьких інтерфейсів та для полегшення розробки односторінкових додатків. Він підтримується Facebook та спільнотою, в створенні брало участь більше тисячі різних розробників.

React намагається допомогти розробникам створювати веб-додатки, які використовують дані, які постійно змінюються. Його мета – бути простим, декларативним та легко сполучуваним. React обробляє лише інтерфейс користувача в додатку.

Відповідно до служби аналізу Javascript (Javascript), Libscore, React в даний час використовується на головних сторінках Imgur, Bleacher Report, Feedly, Airbnb, SeatGeek, HelloSign та інших.

React підтримує віртуальний DOM, замість того, щоб покладатися виключно на DOM браузера. Це дозволяє бібліотеці визначати, які частини DOM були змінені, зіставляючи вміст між новою версією і тим, що зберігається у віртуальному DOM, і

використовувати результат для визначення більш ефективного оновлення браузерного DOM.

Компоненти у React мають властивості (props), та стан (state). Властивості передаються у компонент з батьківського компонента і не можуть бути зміненими у середині компоненту, стан знаходиться у самому компоненту, та може бути змінений за допомогою спеціальної функції `setState`, яка визиває перерендер необхідної частини DOM.

Життєвий цикл – це ряд станів, через які проходить кожен компонент. Вони можуть бути класифіковані на три етапи: збірка, оновлення та знищення. Зазначені стадії мають відповідність у різних методах.

`shouldComponentUpdate` дозволяє розробнику запобігати непотрібному перевизначенню компонента, повертаючи `false`, якщо це не потрібно.

`componentDidMount` викликається після того, як компонент "змонтований" (компонент створено в інтерфейсі користувача, зазвичай асоціюється з вузлом DOM). Це зазвичай використовується для запуску завантаження даних з віддаленого джерела через API.

Компонент `componentWillUnmount` викликається безпосередньо перед тим, як компонент буде "відключено".

Візуалізація є найважливішим методом життєвих циклів і єдиним необхідним для будь-якого компонента. Зазвичай він називається, коли стан компонента оновлюється, відображаючи зміни в інтерфейсі користувача.

React використовує синтаксис, схожий на HTML, який називається JSX. JSX не є необхідним, однак він робить написання коду більш зрозумілим та простим.

3.8 Формальна мова CSS

CSS (Cascading Style Sheets) – це мова графічного дизайну для визначення та створення вигляду структурованого документа, написаного на мові розмітки. За допомогою CSS можна встановлювати візуальний дизайн веб-документів, а також інтерфейси користувача, написані на HTML або XHTML, CSS може бути застосовано до будь-якого документа XML, включаючи XHTML, SVG, XUL, RSS тощо.

CSS, як і HTML та JavaScript, використовується багатьма веб-сайтами для створення візуально привабливих користувацьких інтерфейсів для веб-додатків і графічних інтерфейсів для багатьох мобільних додатків.

CSS розроблений насамперед для позначення поділу вмісту документа та форми вигляду, таких як кольори або шрифти. Це розділення прагне поліпшити доступність документа для рядового користувача, забезпечити більшу гнучкість і контроль у специфікації презентаційних характеристик, дозволяють декільком HTML-документам поділяти один і той же стиль, використовуючи єдину таблицю стилів, розділену у файлі .css, і зменшити складність і повторення коду в структурі документа.

Поділ формату та контенту дає можливість представити той самий документ, позначений у різних стилях, для різних методів візуалізації, наприклад, на екрані, у друкованому вигляді. Веб-сторінка також може відображатися по-різному в залежності від розміру екрана або типу пристрою.

Специфікація CSS описує схему пріоритету для визначення, які правила стилю застосовуються, якщо для конкретного елемента відповідає більше одного правила. Ці правила застосовуються з системою, що називається каскадом, так що пріоритети розраховуються і призначаються правилам, тому результати є передбачуваними.

CSS має простий синтаксис і використовує набір ключових слів англійською мовою, щоб вказати назви різних властивостей стилю.

Таблиця стилів складається з ряду правил. Кожне правило або набір правил складається з одного або більше селекторів і блоку оголошень.

3.9 Висновки до розділу

В даному розділі були описані усі використані технології при розробці веб-додатку. Усі описані технології є зручними, крос-платформними та легко збираються в купу, через що і були обрані на стадії планування розробки веб-додатку.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Опис архітектури проекту

4.1.1 Архітектурний шаблон проектування MVC

Model-view-controller (MVC) – це модель архітектури програмного забезпечення, яка відокремлює дані та бізнес-логіку програми від її подання та модуля, що відповідає за управління подіями та комунікаціями. З цією метою MVC пропонує конструювання трьох різних компонентів, які є моделлю, видом і контролером, тобто, з одного боку, він визначає компоненти для подання інформації, а з іншого боку - для взаємодії користувача. Архітектура програмного забезпечення базується на ідеях повторного використання коду та поділу понять, характеристик, які прагнуть полегшити завдання розробки додатків та їх подальшого обслуговування. (рисунок 4.1).

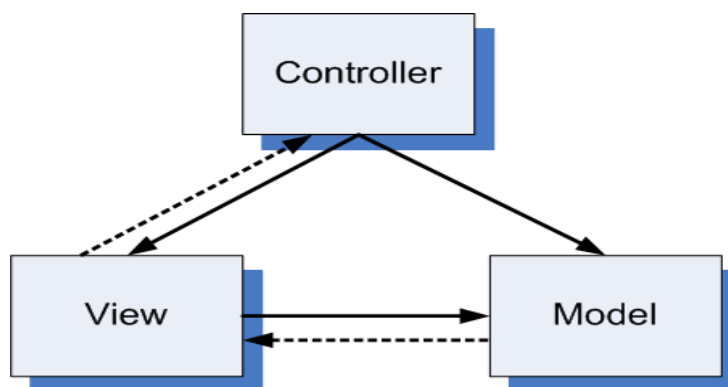


Рисунок 4.1 – Діаграма взаємодії компонентів шаблону MVC

- Model (модель) – модель являє собою дані, якими оперує додаток. Вона надає дані для View, а також реагує на запити від контролера, змінюючи свій стан;
- View (подання) – являє собою компонент веб-додатку для відображення стану моделі в зрозумілому вигляді. У випадку даного веб-додатку це react-сторінки. Подання не змінює дані безпосередньо, дані змінюються за допомогою контролера;
- Controller (контролер) – засіб, за допомогою якого користувач взаємодіє з системою, а також є керуючим елементом для обміну даними між моделлю та поданням.

4.1.2 Архітектура веб-системи

Поняття Onion Architecture (архітектура цибулі) був запропонований Джеффі Палермо в 2008 році, через декілька років ця концепція стала однією з найбільш вживаних типів архітектури при побудові програми з використанням технології ASP.NET Core.

Onion-архітектура (рисунок 4.2) являє собою поділ програми на декілька рівнів. Один з рівнів є незалежним, він знаходиться в самому центрі архітектури. Від цього рівня залежить наступний рівень, від наступного – той що йде після нього і так далі. Тобто навколо першого незалежного рівня нашаровується другий, який залежить від нього. Навколо другого нашаровується третій, який також може залежати і від першого. Образно це може бути виражено у вигляді цибулини, в якій також є ядро, навколо якого нашаровуються всі інші ряди.

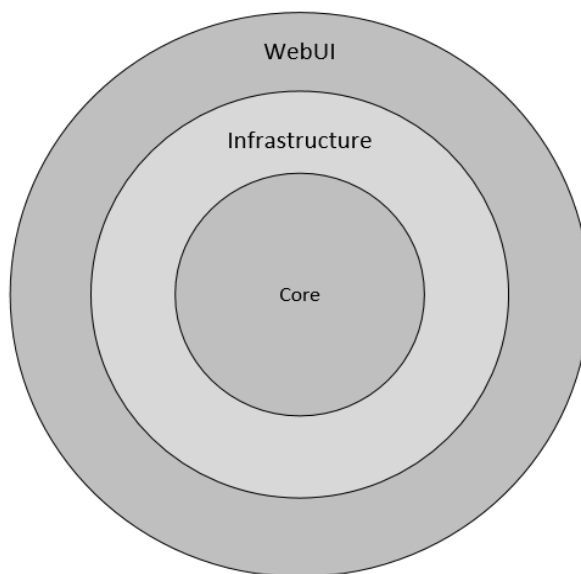


Рисунок 4.2 – Схема onion-архітектури

Першим рівень є ядро (core), яке утворює інтерфейси, які керують роботою з моделлю доменну. Зазвичай це інтерфейси репозиторіїв, через які користувач може взаємодіяти з базою даних. Також цей рівень містить класи таблиць бази даних, які використовуються для взаємодії веб-додатку із базою даних MsSQL за допомогою Entity Framework Core.

Другий рівень – інфраструктура (infrastructure) представляє ті компоненти, які часто подаються змінам. Зазвичай цей рівень утворює користувацький інтерфейс, тести, допоміжні класи, інфраструктури, додатки. До цього рівня також належать конкретні реалізації інтерфейсів, які були оголошені на нижчих рівнях. Наприклад, реалізація інтерфейсу бази даних, яка була оголошена на рівні ядра. Взагалі всі внутрішні рівні, які можна об'єднати в Application Core, визначають тільки інтерфейси, а конкретна реалізація цих інтерфейсів знаходиться на зовнішньому рівні.

Третій рівень, який є останнім містить у собі класи для взаємодії користувача з веб-системою. До нього входять моделі представлення ViewModel, контролери, в яких міститься вся бізнес-логіка, сценарії користувача та css файли, в яких містяться стилі html сторінок.

4.1.3 Концепт інверсії управління

Інверсія управління (Inversion of Control, IoC) – важливий принцип об'єктно-орієнтованого програмування, який використовують для зменшення зв'язності в комп'ютерних програмах. Модулі верхнього рівня не повинні залежати від модулів нижнього рівня. Обидва модулі повинні залежати від абстракції.

Абстракції не повинні залежати від деталей. Деталі повинні залежати від абстракцій.

Однією з реалізацій інверсії управління є внесення залежності (Dependency Injection, DI). DI використовується в багатьох фреймворках, які називаються контейнерами. Наприклад, якщо об'єкт *x* (класу *X*) викликає методи об'єкту *y* (класу *Y*), то *X* залежить від *Y*. Залежність може бути звернена створенням третього класу, а саме класу інтерфейсу *I*, який повинен містити в собі усі методи, які *x* може викликати у об'єкта *y*. Крім того, *Y* повинен реалізовувати інтерфейс *I*. *X* та *Y* наразі обидва залежать від *I*, і клас *X* більше не залежить від класу *Y*; передбачається, що *X* не реалізує *I*.

Це виключення залежності класу *X* від *Y* шляхом створення інтерфейсу *I* і називається Inversion of Control.

Слід сказати, що *Y* може залежати від інших класів. До внесення змін *X* залежав від *Y*, тоді *X* побічно залежав від усіх класів, від яких залежить *Y*. За допомогою застосування Inversion of Control всі побічні залежності були розірвані. При розробці програмного забезпечення, інверсія управління (IoC) є технологією програмування, яка виражена в термінах об'єктно-орієнтованого програмування, в якій співставлення об'єктів виконується під час виконання програми, та зазвичай не відоме під час компіляції. У традиційному програмуванні, потік бізнес-логіки визначається об'єктами, які статично поєднані один з одним. З інверсією управління, потік залежить від графа об'єктів, що створюється збирачем. Це стало можливим завдяки

визначенню поведінки об'єкта через абстракції. Процес зв'язування досягається шляхом внесення залежностей (DI) рисунок (4.3).

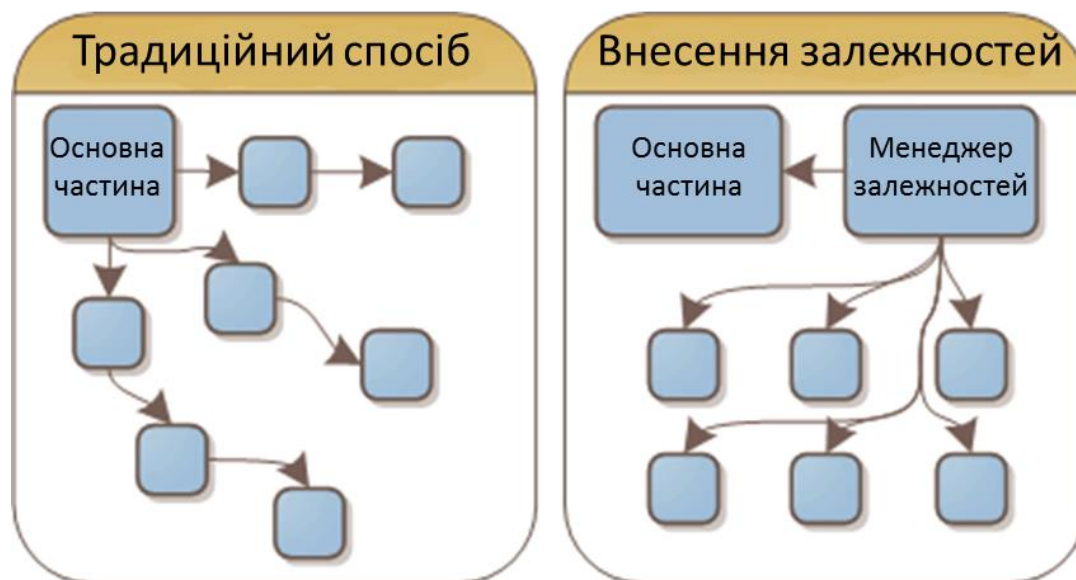


Рисунок 4.3 – Схема зв'язку в системі організованій традиційно та з використанням внесення залежностей

На практиці інверсія управління є стиль архітектурного проектування, де повторне використання коду вирішує проблему черезмірної зв'язності системи. Інверсія управління як шаблон проектування має наступні переваги:

- зменшується зв'язність між модулями;
- кожен модуль може зосередитися на тому, для чого він призначений;
- модулі не мають ніякої інформації про те, як працюють інші системи, вони знають лише інтерфейси;
- заміна одного модулів не впливає на інші модулі.

4.2 Опис структури проекту

Розроблена система представляє собою крос-браузерний та крос-платформний веб-застосунок, який складається з бази даних, в якій зберігається уся необхідна інформація, серверної частини, через яку пролягає взаємодія користувача з базою даних та клієнтської частини, яка відповідає за інтерфейс користувача та розрахунки.

Структура бази даних показана на рисунку 4.4.

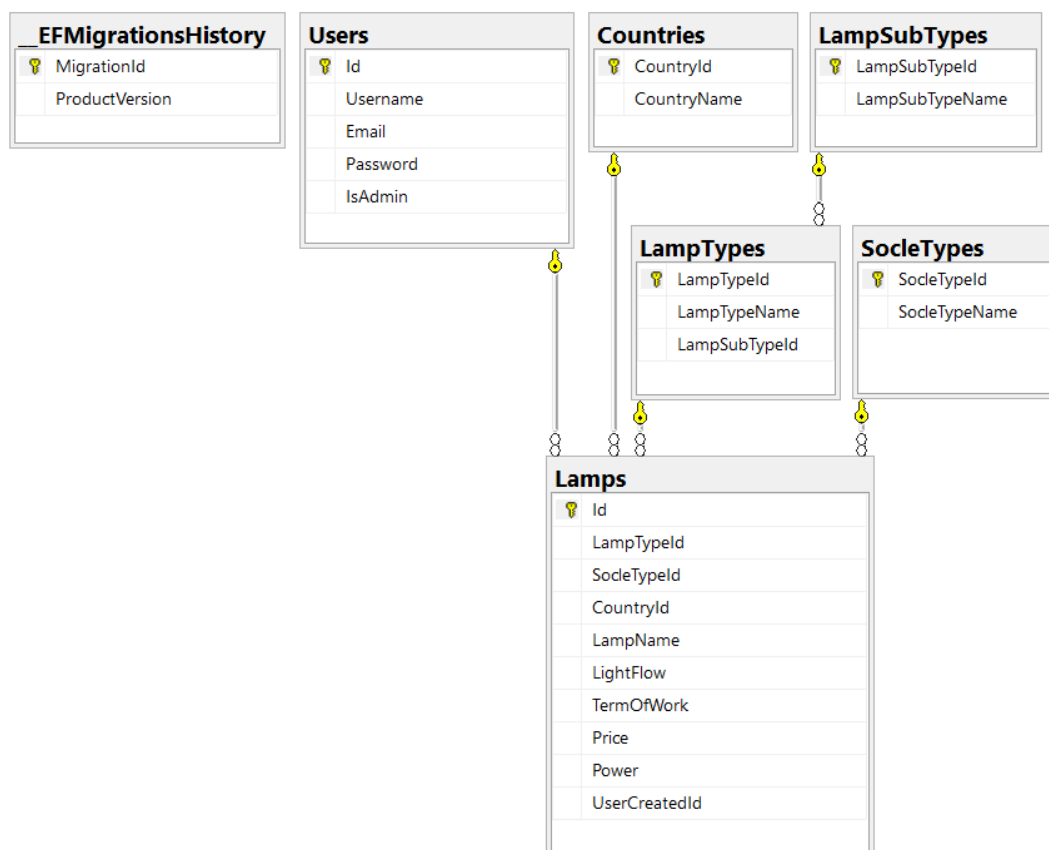


Рисунок 4.4 – Структура бази даних

База даних була створена за допомогою Entity framework code first.

База даних складається з 7 таблиць, одна з яких (_EFMigrationsHistory) була автоматично згенерована Entity Framework Core й містить в собі історію змін в базі

даних та немає відношення до теми роботи. Усі інші, окрім таблиці Users, мають безпосереднє відношення до теми роботи адже зберігають у собі усю необхідну інформацію по лампочкам. Таблиця Users містить в собі інформацію про усіх зареєстрованих користувачів веб-додатку, а саме порядковий номер, пошту, ім'я, пароль та статус користувача, також ця таблиця зв'язана з таблицею Lamps за полем з порядковим номером користувача, це було зроблено задля можливості ідентифікації який саме користувач створив лампочку.

Сервер реалізує багатoshарову архітектуру веб-додатку: шар уявлення, шар доступу до даних, шар Core в якому реалізовані класи-моделі предметної області.

В шарі подання є два контролера AuthController і LampsDefaultController через які здійснюватиметься взаємодія клієнтської частини з сервером. Контроллери реалізують API, які надають доступ до методів доступу до бази даних та дають можливість читати, створювати, редагувати, та видаляти записи відповідно до прав користувача, все це відбувається через аутентифіковані HTTP запити, в разі якщо користувач авторизован. До кожного запиту прикріплюється токен доступу який валідується на сервері по ключу і перевіряється на термін придатності. Якщо користувач вийшов то треба авторизуватись знову. Токен зберігається в LocalStorage при реєстрації або авторизації користувача.

Доступ до даних проходить через Entity framework Core за допомогою LINQ-запитів.

Клієнтська частина взаємодіє з базою даних через API, виконує основні розрахунки, а саме: рахує вартість використання системи освітлення за формулою (2.5), необхідний світловий потік за формулою (2.6) та рахує необхідну кількість ламп відносно отриманого світлового потоку, відображає стилізований інтерфейс користувача та надає користувачу можливість взаємодіяти з елементами інтерфейсу, такими як кнопки переходу в наступний розділ або взаємодіяти з отриманими даними у вигляді чуйного графіку, який дозволяє змінювати вхідні параметри та його тип на льоту завдяки динамічності обраних технологій для сторони клієнта.

При розробці клієнтської частини було використано підхід ціллю якого є якнайменше кількість компонентів, тобто один компонент може використовуватися декілька разів або відображати різні дані в залежності від різних вхідних даних.

4.3 Висновки до розділу

В даному розділі було чітко описано усі використанні архітектури, та детально розписано програмну реалізацію завдання, а саме: структуру бази даних, функції серверу та можливості клієнта.

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

5.1 Інсталяція та системні вимоги

Оскільки зроблена програма є веб-додатком, вона повинна бути завантажена на віддалений хостинг.

Вимоги до хостингу:

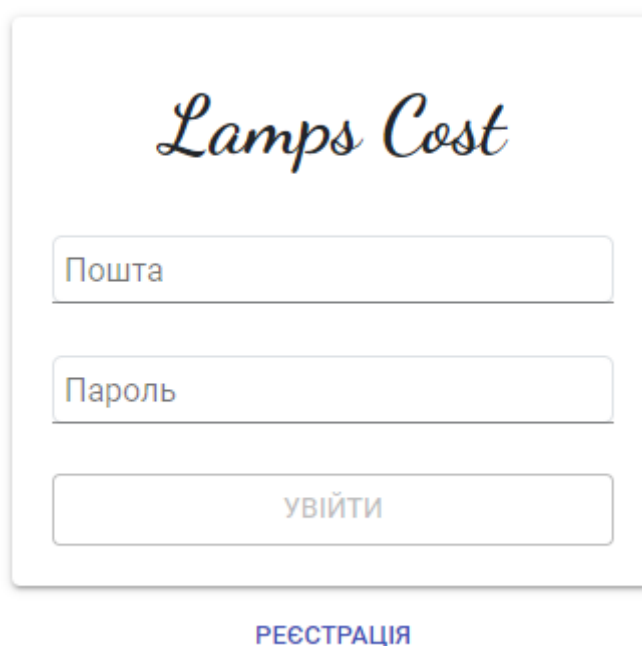
- від 1 гб оперативної пам'яті;
- MsSQL Server 2014 ;
- IIS Server 6.0 або вище;
- Microsoft .Net Framework 4.5;
- операційна система Microsoft Windows Server 2008 і вище.

Щоб завантажити веб-додаток на сервер, необхідно дотримуватись наступних інструкцій:

1. Знайти компанію яка надає послуги хостингу.
2. Вибрати бажаний пакет послуг який буде включати хоча би мінімальні (вище наведені) вимоги до хостингу.
3. Придумати доменне ім'я і замовити реєстрацію домену у будь-якого реєстратора, який підходить за ціною.
4. Скопіювати всі файли на хостинг сервер.
5. Налаштувати і запустити IIS сервер.
6. Додати ваш сайт до списку сайтів у IIS сервері.
7. Направити сайт на ДНС хостингової компанії з боку реєстратора.
8. Зачекати оновлення ДНС з боку реєстратора.

5.2 Робота користувача з системою

При заході на сайт неавторизований користувач бачить вікно авторизації (рисунок 5.1).



Lamps Cost

Пошта

Пароль

УВІЙТИ

РЕЄСТРАЦІЯ

Рисунок 5.1 – Вікно авторизації/реєстрації користувача

Якщо користувач має профіль на сайті, йому необхідно ввести електронну пошту та пароль та натиснути на кнопку увійти, якщо користувач не має профілю, йому необхідно пройти нескладну реєстрацію натиснувши на кнопку «реєстрація»

Після успішної авторизації користувач потрапляє на сторінку з вибором лампочок (рисунок 5.2), де він може передивитись увесь список доступних лампочок та обрати бажані лампочки для подальшого підрахунку за допомогою фільтрів за типом цоколю, країни виробника та діапазоном світлового потоку, також користувач має можливість видаляти небажані лампочки з відфільтрованого списку за допомогою кліку.

Список лампочок
Калькулятор світлового потоку
Калькулятор вартості використання
Редактор лампочок

Фільтри
Тип цоколя
E40
R7S
E27
E14
G13

Країна
Україна
Китай
Туреччина
Америка
Росія
ЄС

Світловий потік
4500 — 5000

Перейти до підрахунку необхідного світлового потоку

| Назва | Тип | Тип цоколя | Потужність | Світловий потік | Строк роботи | Ціна | Країна |
|-------------|--------------------|------------|------------|-----------------|--------------|--------|-----------|
| DELUX GYZ | Ртутно-вольфрамова | E27 | 250 | 4900 | 3000 | 76.8 | Китай |
| Lezard T140 | Люмінесцентна | E27 | 60 | 4700 | 30000 | 398.88 | Туреччина |

Рисунок 5.2 – Розділ вибору лампочок

Після вибору бажаних до перевірки лампочок користувач може перейти до підрахунку необхідно світлового потоку за допомогою кнопки «Перейти до підрахунку необхідного світлового потоку» та перейти до наступного розділу, однак якщо користувач не знайшов бажані до перевірки лампочки, то він має можливість додати власні лампочки перейшовши до розділу «Редактор лампочок» (рисунок 5.6) за посиланням з шапки сайту.

У розділі калькулятора світлового потоку (рисунок 5.3) користувач має можливість підрахувати необхідний світловий потік для кожного з типу ламп заповнивши всі поля, найбільш незрозумілі поля оснащенні підказками, які відображаються при наведенні на іконку зі знаком питання й містять в собі опис та посилання на ГОСТи.

Список лампочек
Калькулятор светового потока
Калькулятор стоимости использования
Редактор лампочек

| | |
|-------------------------------------|--------------------------------------|
| Довжина приміщення | 4 |
| Ширина приміщення | 8 |
| Висота приміщення | 2 |
| Висота робочої поверхні | 0,8 |
| Висота підвісу світильника | 0 |
| Коефіцієнт відображення | 70 50 30 |
| Коефіцієнт запасу | Цементні заводи, ливарні цехи і т.п. |
| Коефіцієнт мінімального освітлення | 1,1 |
| Необхідна освітленість в приміщенні | 300 |

Необхідний світловий потік для газорозрядного типу ламп: 39111 лм
Необхідний світловий потік для ламп розжарення: 33244 лм
Необхідний світловий потік для світлодіодного типу ламп: 29333 лм

Перейти до підрахунку вартості використання

Рисунок 5.3 – Калькулятор світлового потоку

Після підрахунку необхідного світлового потоку користувач може перейти до розрахунку вартості використання лампочок натиснувши на кнопку «Перейти до підрахунку вартості використання»

У розділі підрахунку вартості використання користувач має можливість побачити таблицю (рисунок 5.4) з усіма обраними лампочками та їх порашовану необхідну кількість, яка може бути змінена.

Список лампочек
Калькулятор светового потока
Калькулятор стоимости использования
Редактор лампочек

| Назва | Тип | Тип цоколя | Потужність | Світловий потік | Строк роботи | Ціна | Країна | Кількість |
|---------------|---------------|------------|------------|-----------------|--------------|--------|--------|-----------|
| ЛОН 230-500 | Розжарювання | E40 | 500 | 4800 | 1000 | 30 | Росія | 7 |
| DELUX J-TYPE | Галогенна | R7S | 300 | 4800 | 5000 | 13 | Китай | 7 |
| Feron LB-65 | Люмінесцентна | E14 | 60 | 5000 | 30000 | 469.98 | Китай | 8 |
| Osram L21-840 | Світлодіодна | G13 | 58 | 4600 | 20000 | 50 | Китай | 7 |
| Feron LB-246 | Світлодіодна | G13 | 18 | 1500 | 25000 | 72 | Китай | 20 |
| PHILIPS SON-T | Натрієва | E40 | 150 | 15000 | 28000 | 190.5 | ЄС | 3 |

Рисунок 5.4 – Таблиця з обраними лампочками та їх необхідною кількістю

Під таблицею за лампочками знаходиться розділ з графіком (рисунок 5.5), знизу знаходиться сам графік, на якому показано порівняння обраних лампочок, якщо кількість лампочок перевищує 65 користувачу буде запропоновано автоматично видалити найбільш не вигідні лампочки. Вісь ординати – це вартість використання, а

вісь абсцис відповідає за параметр відносно якого проводиться розрахунок та залежить від вказаного типу графіку. Користувач може регулювати параметри, за яким проводяться розрахунки, а також змінювати тип графіку.

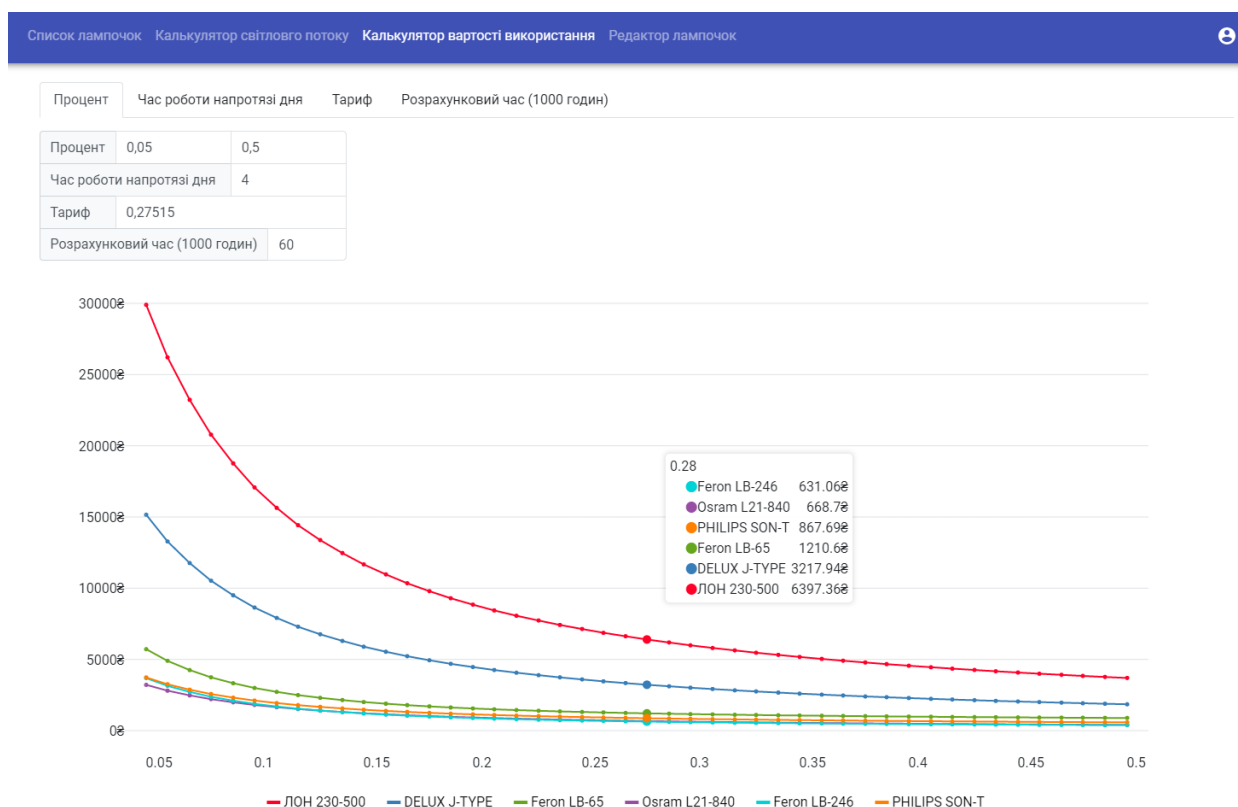


Рисунок 5.5 – Графічне відображення отриманих результатів

Як вже згадувалось раніше, у користувача є можливість додавати нові лампочки у розділі «Редактор лампочок» (рисунок 5.6).

| Список лампочек Калькулятор светового потока Калькулятор стоимости использования Редактор лампочек | | | | | | | | | | |
|----------------------------------------------------------------------------------------------------|---------------|--------------------|------------|------------|-----------------|--------------|--------|-----------|----------|----------|
| Лампа Тип Тип цоколя Страна | | | | | | | | | | |
| Номер | Назва | Тип | Тип цоколя | Потужність | Світловий потік | Строк роботи | Ціна | Країна | | |
| 1 | ЛОН 230-500 | Розжарювання | E40 | 500 | 4800 | 1000 | 30 | Росія | Зберегти | Видалити |
| 2 | DELUX J-TYPE | Галогенна | R7S | 300 | 4800 | 5000 | 13 | Китай | Зберегти | Видалити |
| 3 | DELUX GYZ | Ртутно-вольфрамова | E27 | 250 | 4900 | 3000 | 76,8 | Китай | Зберегти | Видалити |
| 4 | Feron LB-65 | Люмінесцентна | E14 | 60 | 5000 | 30000 | 469,98 | Китай | Зберегти | Видалити |
| 5 | Lezard T140 | Люмінесцентна | E27 | 60 | 4700 | 30000 | 398,88 | Туреччина | Зберегти | Видалити |
| 6 | Osram L21-84 | Світлодіодна | G13 | 58 | 4600 | 20000 | 50 | Китай | Зберегти | Видалити |
| 10 | Feron LB-246 | Світлодіодна | G13 | 18 | 1500 | 25000 | 72 | Китай | Зберегти | Видалити |
| 11 | ДШ 235-40 | Розжарювання | E27 | 40 | 390 | 1000 | 43,8 | Росія | Зберегти | Видалити |
| 12 | PHILIPS SON-T | Натрієва | E40 | 150 | 15000 | 28000 | 190,5 | ЄС | Зберегти | Видалити |
| # | | | | | | | | | Додати | |
| « 1 2 3 ... 15 » | | | | | | | | | | |

Рисунок 5.6 – Редактор лампочок

Лампочки у цьому розділі відображаються відносно прав користувача, адміністратор може додавати, редагувати та видаляти усі наявні лампочки, тип лампочок, країни виробників та тип цоколя. Звичайні користувачі можуть додавати тільки лампочки, а також можуть редагувати та видаляти лампочки, які вони добавили самі.

5.3 Висновки до розділу

У цьому розділі було описано взаємодію користувача зі створеним веб-додатком.

Розроблений веб-застосунок дозволяє підрахувати витрати на освітлення з можливістю вибору лампочок за критеріями, підрахунку необхідної кількості лампочок та виводу отриманої інформації у графічному виді.

ВИСНОВКИ

У роботі була детально розглянута задача порядку розроблення техніко-економічного обґрунтування систем освітлення для промислових споживачів.

Розроблено веб-додаток що надає наступні можливості:

- а) Реєстрація та авторизація користувача;
- б) Вибір лампочки для розрахунку з наявного списку;
- в) Підрахунок необхідного світлового потоку для заданого приміщення;
- г) Підрахунок необхідної кількості ламп залежно від отриманого світлового потоку;
- д) Підрахунок вартості використання лампи відносно часу роботи на добу, відсотку вартості грошей, тарифу та часу за який проводяться розрахунки;
- е) Графічне зображення отриманих результатів;
- ж) Можливість редагування, видалення та додавання лампочок згідно з правами користувача.

Використання клієнтського застосунка та веб-орієнтованість забезпечили високу доступність програмного засобу для будь-якого користувача та можливість запровадити єдиний інтерфейс для взаємодії з продуктом, незалежно від пристрою та програмного забезпечення користувача.

ПЕРЕЛІК ПОСИЛАНЬ

1. User Datagram Protocol [Electronic resource]. — Access mode: <https://technet.microsoft.com/en-us/library/cc785220>.
2. Learn About ASP.NET MVC [Electronic resource]. — Access mode: <http://www.asp.net/mvc>.
3. Visual Studio [Electronic resource]. — Access mode: <https://msdn.microsoft.com/ru-ru/library/52f3sw5c.aspx>.
4. Microsoft Visual Studio 2015 [Electronic resource]. — Access mode: <http://www.pcweek.com/infrastructure/article/detail.php?ID=173068>
5. Pro C# 5.0 and the .NET 4.5 Framework, 6th edition/ Andrew Troelsen — Boston, The Facets of Ruby Series, 2013. — P.456
6. Common Type System [Electronic resource]. — Access mode: [https://msdn.microsoft.com/en-us/library/zcx1eb1e\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/zcx1eb1e(v=vs.110).aspx).
7. ASP.NET и Visual Studio для веб-разработки [Электронный ресурс] — Режим доступа: <https://msdn.microsoft.com/ru-ru/library/dd566231.aspx>.
8. SQL Server 2014 Express edition [Electronic resource]. — Access mode: <https://www.microsoft.com/en-us/server-cloud/products/sql-server>.
9. Tutorial: Writing Transact-SQL Statements [Electronic resource]. — Access mode: <https://msdn.microsoft.com/en-us/library/ms365303.aspx>.
10. JavaScript: The Definitive Guide, 6th Edition/ David Flanagan. // IEEE O'Reilly Media. — 2011. — № 6. — P.1096.
11. Twitter Bootstrap 3 edition [Electronic resource]. — Access mode: <http://getbootstrap.com>.
12. React в действии / Томас Марк Тиленс. // Питер. — 2016. — P.1096.
13. The Onion Architecture [Electronic resource]. — Access mode: <http://jeffreypalermo.com/blog/the-onion-architecture-part-1/>.

14. Views and ViewModels [Electronic resource]. — Access mode: <http://www.asp.net/mvc/overview/older-versions/mvc-music-store/mvc-store-part-3>.
15. The Repository Pattern [Electronic resource]. — Access mode: <https://msdn.microsoft.com/en-us/library/ff649690.aspx>.
16. ASP.NET Identity 2.0 Security Stamp [Electronic resource]. — Access mode: <http://methoddev.com/blg/let-s-talk-software/328/security-stamp>.
17. IdentityUser Class [Electronic resource]. — Access mode: <https://msdn.microsoft.com/ru-u/library/microsoft.aspnet.identity.aspx>.
18. Implementing the Repository and Unit of Work Patterns in an ASP.NET MVC Application [Electronic resource]. — Access mode: <http://www.asp.net/mvc/overview/older-versions/getting-started-with-ef-5>.
19. Економіка.Інструкція з використання / Ха - Юн Чанг. // Наш Формат. — 2014. — Р.322.

Додаток 1

Програмний агент управління та моніторингу вітроенергетичної установки

Специфікація

УКР.НТУУ“КПІ”.ТМ51105_19Б

Аркушів 2

2019

| Позначення | Найменування | Примітки |
|---------------------------------------------------------|-----------------------------|-------------------------|
| Документація | | |
| УКР.НТУУ«КПІ ім. Ігоря Сікорського».ТМ51105_19Б 81-1 | Записка | Пояснювальна записка |
| Компоненти | | |
| УКР.НТУУ«КПІ». ТМ51105_19Б 12-1 | Текст програмного модулю | |
| УКР.НТУУ«КПІ». ТМ51105_19Б 13-1 | Опис програми | |

Додаток 2

Програмний агент управління та моніторингу вітроенергетичної установки

Текст програмного модулю

УКР.НТУУ“КПІ”.ТМ51105_19Б 12-1

Аркушів 10

2019

```

import React, { Component } from 'react';
import { Graph } from './Graph/Graph';
import { Inputs } from './Inputs';
import { TypeChose } from './TypeChose';
import { types } from '../constants/types';
export class GraphContainer extends Component {
  constructor(props) {
    super(props);
    this.state = {
      ...Object.fromEntries(types.map(item => [item.key, item.value])),
      type: types[0].key,
      range: Object.fromEntries(types.map(item => [item.key,
item.range])),
    };
    this.handleParamChange = this.handleParamChange.bind(this);
    this.handleRangeChange = this.handleRangeChange.bind(this);
    this.handleTypeChange = this.handleTypeChange.bind(this);
  }
  handleRangeChange(pos, el) {
    const range = { ...this.state.range };
    range[this.state.type][pos] = el;
    this.setState(() => ({ range }));
  }
  handleTypeChange(type) {
    this.setState(() => ({ type }));
  }
  handleParamChange(param, value) {
    this.setState(() => ({ [param]: value }));
  }
  getDataList() {
    const { range, type } = this.state;
    const { amt, products } = this.props;

    if (type === types[1].key)
      return ((percent, whpd, tariff, wh) =>
        products.map((product, index) => {
          const data = [];
          for (let i = whpd[0]; i <= whpd[1]; i++) {
            const i1000 = Math.pow(1 + percent, 600 / (219 *
i)) - 1;

            let sum = product.price;
            for (let j = 1; j <= wh; j++) {
              const price = j !== wh && j %
(product.termOfWork / 1000) !== 0 ? 0 : product.price * amt[index];
              sum += (price + tariff * amt[index] *
product.power) / Math.pow(1 + i1000, j);
            }
            data.push({ x: Math.round(i * 100) / 100, y:
Math.round(sum * 100) / 100 });
          }
          return data;
        })
      )(this.state[types[0].key], range[type], this.state[types[2].key],
this.state[types[3].key]);

    if (type === types[2].key)
      return ((percent, whpd, tariff, wh) => {
        const i1000 = Math.pow(1 + percent, 600 / (219 * whpd)) - 1;
        return products.map((product, index) => {
          const data = [];

```

```

        for (let i = tariff[0]; i < tariff[1] + 0.000001; i +=
0.01) {
            let sum = product.price;
            for (let j = 1; j <= wh; j++) {
                const price = j !== wh && j %
(product.termOfWork / 1000) !== 0 ? 0 : product.price * amt[index];
                sum += (price + i * amt[index] *
product.power) / Math.pow(1 + i1000, j);
            }
            data.push({ x: Math.round(i * 100) / 100, y:
Math.round(sum * 100) / 100 });
        }
        return data;
    })
    )(this.state[types[0].key], this.state[types[1].key], range[type],
this.state[types[3].key]);

    if (type === types[0].key)
        return ((percent, whpd, tariff, wh) =>
            products.map((product, index) => {
                const data = [];
                for (let i = percent[0]; i < percent[1] + 0.000001; i
+= 0.01) {
                    const i1000 = Math.pow(1 + i, 600 / (219 * whpd))
- 1;

                    let sum = product.price;
                    for (let j = 1; j <= wh; j++) {
                        const price = (j !== wh && j %
(product.termOfWork / 1000) !== 0) ? 0 : product.price * amt[index];
                        sum += (price + tariff * amt[index] *
product.power) / Math.pow(1 + i1000, j);
                    }
                    data.push({ x: Math.round(i * 100) / 100, y:
Math.round(sum * 100) / 100 });
                }
                return data;
            })
        )(range[type], this.state[types[1].key], this.state[types[2].key],
this.state[types[3].key]);

    return ((percent, whpd, tariff, wh) =>
        products.map((product, index) => {
            const i1000 = Math.pow(1 + percent, 600 / (219 * whpd)) - 1;
            const data = [];
            for (let i = wh[0]; i <= wh[1]; i++) {
                let sum = product.price;
                for (let j = 1; j <= i; j++) {
                    const price = (j !== i && j % (product.termOfWork
/ 1000) !== 0) ? 0 : product.price * amt[index];
                    sum += (price + tariff * amt[index] *
product.power) / Math.pow(1 + i1000, j);
                }
                data.push({ x: i, y: Math.round(sum * 100) / 100 });
            }
            return data;
        })
    )(this.state[types[0].key], this.state[types[1].key],
this.state[types[2].key], range[type]);
}
render() {

```

```

const data = this.getDataList();
const { range, type } = this.state;
const { products } = this.props;
return (
  <div>
    <TypeChose
      types={types}
      onTypeChange={this.handleTypeChange}
      checked={type}
    />
    <Inputs
      range={range}
      type={type}
      types={types}
      params={types.map(item => this.state[item.key])}
      onRangeChange={this.handleRangeChange}
      onParamChange={this.handleParamChange}
    />
    <Graph
      info={products.map(item => item.lampName)}
      data={data}
      range={range[type]}
      lineSelect={Array(data.length).fill(true)}
    />
  </div >
);
}
}

import React, { Component } from 'react';
import { Lines } from './Lines';
import { Axis } from './Axis';
import { Legend } from './Legend';
const palette = require('google-palette');
export class Graph extends Component {
  constructor(props) {
    super(props);
    this.state = {
      lineSelect: props.lineSelect
    };
    this.max = 0;
    this.step = 0;
    this.isUpdated = false
    this.handleLineSelect = this.handleLineSelect.bind(this)
  }
  updateCanvas(max, isNew) {
    const ctx = this._back.getContext('2d');
    const width = this._back.width;
    const height = this._back.height;
    if (isNew) ctx.translate(0, height);
    else ctx.clearRect(0, 0, width, -height);
    const graphStep = this.step / (max / (height - 100));
    ctx.beginPath();
    for (let i = 0; i <= height - 100 + graphStep; i += graphStep) {
      ctx.moveTo(0, -50 - i);
      ctx.lineTo(width, -50 - i);
    }
    ctx.strokeStyle = "#dee2e6";
    ctx.stroke();
  }
}

```

```

componentDidMount() {
  this.updateCanvas(this.max, true);
}
componentWillUpdate() {
  this.isUpdated = !this.isUpdated;
}
componentDidUpdate() {
  this.updateCanvas(this.max, false);
}
generateGraphStep() {
  const data = this.props.data;
  const columOfMax = data[0][0].y > data[0][data[0].length - 1].y ? 0 :
data[0].length - 1;
  let max = data.reduce((max, min) => min[columOfMax].y > max ?
min[columOfMax].y : max, 0);
  let maxSize = Math.pow(10, max.toFixed(0).length - 2);
  let step = maxSize;
  max = Math.ceil(max / maxSize) * maxSize;
  if (maxSize * 20 - max > 0) {
    step = maxSize * 2;
    if (+max.toString()[1] % 2 !== 0) max += maxSize;
  }
  else if (maxSize * 50 - max > 0) {
    step = maxSize * 5;
    let h = +max.toString()[1];
    if (h < 5 && h !== 0) max += maxSize * (5 - h);
    else if (h > 5) max += maxSize * (10 - h);
  }
  else if (maxSize * 100 - max > 0) {
    step = maxSize * 10;
    let h = +max.toString()[1];
    if (h !== 0) max += maxSize * (10 - h);
  }
  this.max = max;
  this.step = step;
}
handleLineSelect(isShow, index) {
  const lineSelect = this.state.lineSelect.slice();
  lineSelect[index] = isShow;
  this.setState(() => ({ lineSelect }));
}
getBackRef = (node) => { this._back = node }
render() {
  this.generateGraphStep();
  const { range, info, data } = this.props;
  const colors = palette('mpn65', data.length);
  const { lineSelect } = this.state;
  return (
    <div className="graph">
      <canvas ref={this.getBackRef} width={1200} height={600} />
      <Lines
        data={data} info={info} max={this.max}
        colors={colors} lineSelect={lineSelect}
        isUpdated={this.isUpdated}
      />
      <Axis
        max={this.max}
        step={this.step}
        x={range}
      />
    </div>
  );
}

```



```

        <Legend
          info={info} colors={colors}
          onlineSelect={this.handleLineSelect}
        />
      </div>
    );
  }
}

import React, { Component } from 'react';

export class Lines extends Component {
  constructor(props) {
    super(props);
    this.state = {};
    this.dots = [];
  }
  getDots(x, y) {
    const dots = this.dots;
    const pos = dots.reduceRight((prev, cur, index) => cur.x.pos >= x ? index
: prev, dots.length - 1);
    if (this.state.pos !== pos || this.state.pos === undefined) {
      const colors = this.props.colors;
      const ctx = this._dots.getContext('2d');
      const width = this._lines.width;
      const height = this._lines.height;
      ctx.clearRect(0, 0, width, height);
      dots[pos].y.forEach((item, index) => {
        if (item !== null) {
          ctx.beginPath();
          ctx.fillStyle = `#${colors[index]}`;
          ctx.arc(dots[pos].x.pos, item.pos, 5, 0, 2 * Math.PI);
          ctx.fill();
        }
      });
      this.setState(() => ({
        pos,
        x: dots[pos].x,
        y: dots[pos].y
      }));
    }
    if (this._tooltip !== undefined) {
      this._tooltip.style.top = 300 + y + "px";
      this._tooltip.style.left = 380 + x + "px";
    }
  }

  createLines(data, isNew) {
    const ctx = this._lines.getContext('2d');
    const width = this._lines.width;
    const height = this._lines.height;
    const colors = this.props.colors;
    const ky = this.props.max / (height - 100);
    if (isNew) ctx.translate(0, height);
    else ctx.clearRect(0, 0, width, -height);
    ctx.lineWidth = 2;
    const newData = data.map((item, i) => {
      if (!this.props.lineSelect[i])
        return null;
    });
  }
}

```

```

    ctx.beginPath();
    return item.map((el, j) => {
      const pos = {
        x: 25 + Math.floor(j / (item.length - 1) * (width -
50)),
        y: -50 - item[j].y / ky
      };
      ctx.lineTo(pos.x, pos.y);
      ctx.arc(pos.x, pos.y, 1.5, 0, 2 * Math.PI);
      ctx.moveTo(pos.x, pos.y);
      if (j === item.length - 1) {
        ctx.strokeStyle = `#${colors[i]}`;
        ctx.stroke();
      }
      return { ...el, pos };
    });
  });
  this.dots = newData[0].map((el, j) => ({
    x: {
      value: el.x,
      pos: el.pos.x
    },
    y: newData.map(item => item !== null ?
      {
        value: item[j].y,
        pos: height + item[j].pos.y
      }
      :
      null
    )
  }));
}
componentDidMount() {
  this.createLines(this.props.data, true);
}
componentDidUpdate(prevProps) {
  if (this.props.isUpdated !== prevProps.isUpdated)
    this.createLines(this.props.data, false);
}
showTooltip() {
  this.setState(() => ({
    isShow: 'block'
  }));
}
hideTooltip() {
  const ctx = this._dots.getContext('2d');
  const width = this._lines.width;
  const height = this._lines.height;
  ctx.clearRect(0, 0, width, height);
  this.setState(() => ({
    isShow: 'none'
  }));
}
render() {
  return (
    <i>
      <canvas ref={node => { this._lines = node }} width={1200}
height={600} />
    </i>
  );
}

```

```

        this.state.x !== undefined &&
        <div className="graph-tooltip" ref={(node) => {
this._tooltip = node }} style={{ top: 0, left: 0, display: this.state.isShow }}>
            <table>
                {this.state.x.value}
                {
                    this.state.y
                        .map((item, index) => ({ value:
item.value, color: this.props.colors[index], name: this.props.info[index] )))
                        .sort((a, b) => a.value -
b.value)
                        .map((item, index) =>
                            item !== null ?
                                <tr key={index}>
                                    <td style={{
color: `#${item.color}` }}>●</td>

                                    <td>{item.name}</td>

                                    <td>{item.value}</td>

                                </tr>
                                :
                                null
                            )
                        }
                }
            </table>
        </div>
    }
    <canvas
        ref={(node) => { this._dots = node }} width={1200}
height={600}
        onMouseMove={(e) => this.getDots(e.nativeEvent.offsetX,
e.nativeEvent.offsetY)}
        onMouseLeave={() => this.hideTooltip()}
        onMouseEnter={() => this.showTooltip()}
    />
    </i>
    );
}
}

import React, { Component } from 'react';
import { Select } from './Select';
import { Container, Row, Col } from 'react-bootstrap';

import { roomProps } from '../constants/roomProps';
import { nTable } from '../constants/nTable';
class CalcLightFlow extends Component {
    displayName = CalcLightFlow.name
    constructor(props) {
        super(props);
        this.state = {
            ...Object.fromEntries(roomProps.map(item => [item.name,
item.value]))
        };
        this.k = [
            [2, 1.7, 1.5],
            [1.8, 1.5, 1.3],
            [1.5, 1.3, 1.1],

```



```
keys={{
```

```
value: item.name }}

    selected={this.state[item.name]}

    onChange={this.handleChange}

  />
}
</div>
))
}
</div>
<div>Необхідний світловий потік для
газорозрядного типу ламп: {lightFlow[0]} лм</div>
<div>Необхідний світловий потік для ламп
розжарення: {lightFlow[1]} лм</div>
<div>Необхідний світловий потік для світлодіодного
типу ламп: {lightFlow[2]} лм</div>
</Col>
</Row>
<input className="btn btn-primary" type="button"
value="Перейти до підрахунку вартості використання" onClick={this.setAmt(lightFlow)}
/>
</Container>
);
}
}

export default CalcLightFlow;
```

Додаток 3

Програмний агент управління та моніторингу вітроенергетичної установки

Опис програмного модулю

УКР.НТУУ“КПІ”.ТМ51105_19Б 13-1

Аркушів 6

АНОТАЦІЯ

Розроблений веб-додаток техніко-економічного обґрунтування систем освітлення для промислових споживачів може вести підрахунок витрат на освітлення з можливістю вибору лампочок за критеріями, підрахунок необхідної кількості лампочок та виводити отриману інформацію у графічному виді .

Користувачами можуть бути люди, які мають будь-який пристрій з встановленим браузером та доступом до мережі інтернет.

ЗМІСТ

| | |
|------------------------------------------|----|
| 1. Відомості про програмний модуль | 72 |
| 1.1. Опис логічної структури..... | 73 |
| 1.2. Вхідні та вихідні дані..... | 73 |
| 2. Використовувані технічні засоби | 74 |

1. ВІДОМОСТІ ПРО ПРОГРАМНИЙ МОДУЛЬ

При створенні програмного забезпечення були використані такі засоби реалізації:

- Середовище розробки Microsoft Visual Studio, адже має зручний інтерфейс та великий функціонал для розробки серверної частини;
- Середовище розробки Microsoft Visual Studio Code, адже має зручний інтерфейс та великий функціонал для розробки клієнтської частини;
- Мова програмування C# для написання серверної частини додатку;
- Мова програмування JavaScript для написання клієнтської частини додатку;
- Фреймворк .NET Core для організації архітектури додатку;
- Технологія ASP.NET Core для створення серверу;
- Шаблон проектування архітектури додатку Model-View-ViewModel для поділу моделі і її представлення, що необхідно для їх зміни окремо один від одного;
- Технологія Entity Framework Core для доступу до даних бази даних;
- Фреймворк React для написання чуйного інтерфейсу;
- Формальна мова CSS для надання інтерфейсу стилістичної привабливості
- Система контролю версій GitHub для контролю версій розробленої системи;
- Реляційну базу даних Microsoft SQL, вона є безкоштовною; має легкий та зрозумілий веб-інтерфейс та легко інтегрується з середовищем розробки Visual Studio;

1.1. Опис логічної структури

Для реалізації задачі техніко-економічного обґрунтування систем освітлення промислових споживачів був розроблений веб-додаток, який використовує базу даних, що зберігає інформацію про лампочки.

Інтерфейс користувача — виконано з використанням таких технологій: мова програмування JavaScript для написання основної логіки, фреймворк React для створення чуйного інтерфейсу, формальна мова CSS для придання інтерфейсу зрозумілого та красивого виду.

1.2. Вхідні та вихідні дані

Вхідними даними для системи є характеристики лампочок та характеристики приміщення.

Вихідними даними є чуйний графік, що несе у собі інформацію про вартість використання в залежності від обраного типу графіку та параметрів.

2. ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Розроблений веб-додаток було протестовано на персональному комп'ютері, який працює на базі процесору x64 Intel Core i7 (8th Gen) та має 8 Гб оперативної пам'яті. Розроблений веб-додаток базується на відкритому хостингу, що дозволяє запускати його на будь-якому пристрої з встановленим браузером та доступом до мережі інтернет.